Gunicorn Documentation

Release 20.0.4

Benoit Chesneau

Jul 09, 2020

Contents

Website http://gunicorn.org

Source code https://github.com/benoitc/gunicorn

Issue tracker https://github.com/benoitc/gunicorn/issues

IRC `#gunicorn` on Freenode

Usage questions https://github.com/benoitc/gunicorn/issues

Gunicorn 'Green Unicorn' is a Python WSGI HTTP Server for UNIX. It's a pre-fork worker model ported from Ruby's Unicorn project. The Gunicorn server is broadly compatible with various web frameworks, simply implemented, light on server resources, and fairly speedy.

CHAPTER 1

Features

- Natively supports WSGI, Django, and Paster
- Automatic worker process management
- Simple Python configuration
- Multiple worker configurations
- Various server hooks for extensibility
- Compatible with Python 3.x >= 3.5

CHAPTER 2

Contents

2.1 Installation

Requirements Python 3.x >= 3.5

To install the latest released version of Gunicorn:

```
$ pip install gunicorn
```

2.1.1 From Source

You can install Gunicorn from source just as you would install any other Python package:

```
$ pip install git+https://github.com/benoitc/gunicorn.git
```

This will allow you to keep up to date with development on GitHub:

```
$ pip install -U git+https://github.com/benoitc/gunicorn.git
```

2.1.2 Async Workers

You may also want to install Eventlet or Gevent if you expect that your application code may need to pause for extended periods of time during request processing. Check out the design docs for more information on when you'll want to consider one of the alternate worker types.

```
$ pip install greenlet            # Required for both
$ pip install eventlet            # For eventlet workers
$ pip install gunicorn[eventlet]  # Or, using extra
$ pip install gevent              # For gevent workers
$ pip install gunicorn[gevent]    # Or, using extra
```

Note: Both require `greenlet`, which should get installed automatically. If its installation fails, you probably need to install the Python headers. These headers are available in most package managers. On Ubuntu the package name for `apt-get` is `python-dev`.

Gevent also requires that `libevent` 1.4.x or 2.0.4 is installed. This could be a more recent version than what is available in your package manager. If Gevent fails to build even with libevent installed, this is the most likely reason.

2.1.3 Extra Packages

Some Gunicorn options require additional packages. You can use the `[extra]` syntax to install these at the same time as Gunicorn.

Most extra packages are needed for alternate worker types. See the design docs for more information on when you'll want to consider an alternate worker type.

- `gunicorn[eventlet]` - Eventlet-based greenlets workers
- `gunicorn[gevent]` - Gevent-based greenlets workers
- `gunicorn[gthread]` - Threaded workers
- `gunicorn[tornado]` - Tornado-based workers, not recommended

If you are running more than one instance of Gunicorn, the *proc_name* setting will help distinguish between them in tools like `ps` and `top`.

- `gunicorn[setproctitle]` - Enables setting the process name

Multiple extras can be combined, like `pip install gunicorn[gevent,setproctitle]`.

2.1.4 Debian GNU/Linux

If you are using Debian GNU/Linux it is recommended that you use system packages to install Gunicorn except maybe when you want to use different versions of Gunicorn with virtualenv. This has a number of advantages:

- Zero-effort installation: Automatically starts multiple Gunicorn instances based on configurations defined in `/etc/gunicorn.d`.
- Sensible default locations for logs (`/var/log/gunicorn`). Logs can be automatically rotated and compressed using `logrotate`.
- Improved security: Can easily run each Gunicorn instance with a dedicated UNIX user/group.
- Sensible upgrade path: Upgrades to newer versions result in less downtime, handle conflicting changes in configuration options, and can be quickly rolled back in case of incompatibility. The package can also be purged entirely from the system in seconds.

stable ("stretch")

The version of Gunicorn in the Debian "stable" distribution is 19.6.0 (June 2017). You can install it using:

```
$ sudo apt-get install gunicorn
```

You can also use the most recent version by using Debian Backports. First, copy the following line to your `/etc/apt/sources.list`:

```
deb http://ftp.debian.org/debian stretch-backports main
```

Then, update your local package lists:

```
$ sudo apt-get update
```

You can then install the latest version using:

```
$ sudo apt-get -t stretch-backports install gunicorn
```

oldstable ("jessie")

The version of Gunicorn in the Debian "oldstable" distribution is 19.0 (June 2014). You can install it using:

```
$ sudo apt-get install gunicorn
```

You can also use the most recent version by using Debian Backports. First, copy the following line to your `/etc/apt/sources.list`:

```
deb http://ftp.debian.org/debian jessie-backports main
```

Then, update your local package lists:

```
$ sudo apt-get update
```

You can then install the latest version using:

```
$ sudo apt-get -t jessie-backports install gunicorn
```

Testing ("buster") / Unstable ("sid")

"buster" and "sid" contain the latest released version of Gunicorn. You can install it in the usual way:

```
$ sudo apt-get install gunicorn
```

2.1.5 Ubuntu

Ubuntu 12.04 (trusty) or later contains the Gunicorn package by default so that you can install it in the usual way:

```
$ sudo apt-get update
$ sudo apt-get install gunicorn
```

2.2 Running Gunicorn

You can run Gunicorn by using commands or integrate with popular frameworks like Django, Pyramid, or TurboGears. For deploying Gunicorn in production see *Deploying Gunicorn*.

2.2.1 Commands

After installing Gunicorn you will have access to the command line script `gunicorn`.

gunicorn

Basic usage:

```
$ gunicorn [OPTIONS] [WSGI_APP]
```

Where `WSGI_APP` is of the pattern `$(MODULE_NAME):$(VARIABLE_NAME)`. The module name can be a full dotted path. The variable name refers to a WSGI callable that should be found in the specified module.

Changed in version 20.1.0: `WSGI_APP` is optional if it is defined in a *config* file.

Example with the test app:

```
def app(environ, start_response):
    """Simplest possible application object"""
    data = b'Hello, World!\n'
    status = '200 OK'
    response_headers = [
        ('Content-type', 'text/plain'),
        ('Content-Length', str(len(data)))
    ]
    start_response(status, response_headers)
    return iter([data])
```

You can now run the app with the following command:

```
$ gunicorn --workers=2 test:app
```

The variable name can also be a function call. In that case the name will be imported from the module, then called to get the application object. This is commonly referred to as the "application factory" pattern.

```
def create_app():
    app = FrameworkApp()
    ...
    return app
```

```
$ gunicorn --workers=2 'test:create_app()'
```

Positional and keyword arguments can also be passed, but it is recommended to load configuration from environment variables rather than the command line.

Commonly Used Arguments

- `-c CONFIG, --config=CONFIG` - Specify a config file in the form `$(PATH)`, `file:$(PATH)`, or `python:$(MODULE_NAME)`.
- `-b BIND, --bind=BIND` - Specify a server socket to bind. Server sockets can be any of `$(HOST)`, `$(HOST):$(PORT)`, `fd://$(FD)`, or `unix:$(PATH)`. An IP is a valid `$(HOST)`.
- `-w WORKERS, --workers=WORKERS` - The number of worker processes. This number should generally be between 2-4 workers per core in the server. Check the *FAQ* for ideas on tuning this parameter.
- `-k WORKERCLASS, --worker-class=WORKERCLASS` - The type of worker process to run. You'll definitely want to read the production page for the implications of this parameter. You can set this to `$(NAME)` where `$(NAME)` is one of `sync`, `eventlet`, `gevent`, `tornado`, `gthread`. `sync` is the default. See the *worker_class* documentation for more information.

- `-n APP_NAME, --name=APP_NAME` - If setproctitle is installed you can adjust the name of Gunicorn process as they appear in the process system table (which affects tools like `ps` and `top`).

Settings can be specified by using environment variable *GUNICORN_CMD_ARGS*.

See *Configuration Overview* and *Settings* for detailed usage.

2.2.2 Integration

Gunicorn also provides integration for Django and Paste Deploy applications.

Django

Gunicorn will look for a WSGI callable named `application` if not specified. So for a typical Django project, invoking Gunicorn would look like:

```
$ gunicorn myproject.wsgi
```

Note: This requires that your project be on the Python path; the simplest way to ensure that is to run this command from the same directory as your `manage.py` file.

You can use the –env option to set the path to load the settings. In case you need it you can also add your application path to `PYTHONPATH` using the –pythonpath option:

```
$ gunicorn --env DJANGO_SETTINGS_MODULE=myproject.settings myproject.wsgi
```

Paste Deployment

Frameworks such as Pyramid and Turbogears are typically configured using Paste Deployment configuration files. If you would like to use these files with Gunicorn, there are two approaches.

As a server runner, Gunicorn can serve your application using the commands from your framework, such as `pserve` or `gearbox`. To use Gunicorn with these commands, specify it as a server in your configuration file:

```
[server:main]
use = egg:gunicorn#main
host = 127.0.0.1
port = 8080
workers = 3
```

This approach is the quickest way to get started with Gunicorn, but there are some limitations. Gunicorn will have no control over how the application is loaded, so settings such as reload will have no effect and Gunicorn will be unable to hot upgrade a running application. Using the daemon option may confuse your command line tool. Instead, use the built-in support for these features provided by that tool. For example, run `pserve --reload` instead of specifying `reload = True` in the server configuration block. For advanced configuration of Gunicorn, such as Server Hooks specifying a Gunicorn configuration file using the `config` key is supported.

To use the full power of Gunicorn's reloading and hot code upgrades, use the paste option to run your application instead. When used this way, Gunicorn will use the application defined by the PasteDeploy configuration file, but Gunicorn will not use any server configuration defined in the file. Instead, configure gunicorn.

For example:

```
$ gunicorn --paste development.ini -b :8080 --chdir /path/to/project
```

Or use a different application:

```
$ gunicorn --paste development.ini#admin -b :8080 --chdir /path/to/project
```

With both approaches, Gunicorn will use any loggers section found in Paste Deployment configuration file, unless instructed otherwise by specifying additional logging settings.

2.3 Configuration Overview

Gunicorn reads configuration information from five places.

Gunicorn first reads environment variables for some configuration *settings*.

Gunicorn then reads configuration from a framework specific configuration file. Currently this only affects Paster applications.

The third source of configuration information is an optional configuration file `gunicorn.conf.py` searched in the current working directory or specified using a command line argument. Anything specified in this configuration file will override any framework specific settings.

The fourth place of configuration information are command line arguments stored in an environment variable named `GUNICORN_CMD_ARGS`.

Lastly, the command line arguments used to invoke Gunicorn are the final place considered for configuration settings. If an option is specified on the command line, this is the value that will be used.

When a configuration file is specified in the command line arguments and in the `GUNICORN_CMD_ARGS` environment variable, only the configuration file specified on the command line is used.

Once again, in order of least to most authoritative:

1. Environment Variables
2. Framework Settings
3. Configuration File
4. `GUNICORN_CMD_ARGS`
5. Command Line

Note: To print your resolved configuration when using the command line or the configuration file you can run the following command:

```
$ gunicorn --print-config APP_MODULE
```

To check your resolved configuration when using the command line or the configuration file you can run the following command:

```
$ gunicorn --check-config APP_MODULE
```

It also allows you to know if your application can be launched.

2.3.1 Command Line

If an option is specified on the command line, it overrides all other values that may have been specified in the app specific settings, or in the optional configuration file. Not all Gunicorn settings are available to be set from the command line. To see the full list of command line settings you can do the usual:

```
$ gunicorn -h
```

There is also a `--version` flag available to the command line scripts that isn't mentioned in the list of *settings*.

2.3.2 Configuration File

The configuration file should be a valid Python source file with a **python extension** (e.g. *gunicorn.conf.py*). It only needs to be readable from the file system. More specifically, it does not have to be on the module path (sys.path, PYTHONPATH). Any Python is valid. Just consider that this will be run every time you start Gunicorn (including when you signal Gunicorn to reload).

To set a parameter, just assign to it. There's no special syntax. The values you provide will be used for the configuration values.

For instance:

```
import multiprocessing

bind = "127.0.0.1:8000"
workers = multiprocessing.cpu_count() * 2 + 1
```

All the settings are mentioned in the *settings* list.

2.3.3 Framework Settings

Currently, only Paster applications have access to framework specific settings. If you have ideas for providing settings to WSGI applications or pulling information from Django's settings.py feel free to open an issue to let us know.

Paster Applications

In your INI file, you can specify to use Gunicorn as the server like such:

```
[server:main]
use = egg:gunicorn#main
host = 192.168.0.1
port = 80
workers = 2
proc_name = brim
```

Any parameters that Gunicorn knows about will automatically be inserted into the base configuration. Remember that these will be overridden by the config file and/or the command line.

2.4 Settings

This is an exhaustive list of settings for Gunicorn. Some settings are only able to be set from a configuration file. The setting name is what should be used in the configuration file. The command line arguments are listed as well for reference on setting at the command line.

Note: Settings can be specified by using environment variable `GUNICORN_CMD_ARGS`. All available command line arguments can be used. For example, to specify the bind address and number of workers:

```
$ GUNICORN_CMD_ARGS="--bind=127.0.0.1 --workers=3" gunicorn app:app
```

New in version 19.7.

2.4.1 Config File

config

- `-c CONFIG, --config CONFIG`
- `None`

The Gunicorn config file.

A string of the form `PATH`, `file:PATH`, or `python:MODULE_NAME`.

Only has an effect when specified on the command line or as part of an application specific configuration.

Changed in version 19.4: Loading the config from a Python module requires the `python:` prefix.

wsgi_app

- `None`

A WSGI application path in pattern `$(MODULE_NAME):$(VARIABLE_NAME)`.

New in version 20.1.0.

2.4.2 Debugging

reload

- `--reload`
- `False`

Restart workers when code changes.

This setting is intended for development. It will cause workers to be restarted whenever application code changes.

The reloader is incompatible with application preloading. When using a paste configuration be sure that the server block does not import any application code or the reload will not work as designed.

The default behavior is to attempt inotify with a fallback to file system polling. Generally, inotify should be preferred if available because it consumes less system resources.

Note: In order to use the inotify reloader, you must have the `inotify` package installed.

reload_engine

- `--reload-engine STRING`
- `auto`

The implementation that should be used to power *reload*.

Valid engines are:

- 'auto'
- 'poll'
- 'inotify' (requires inotify)

New in version 19.7.

reload_extra_files

- `--reload-extra-file FILES`
- `[]`

Extends *reload* option to also watch and reload on additional files (e.g., templates, configurations, specifications, etc.).

New in version 19.8.

spew

- `--spew`
- `False`

Install a trace function that spews every line executed by the server.

This is the nuclear option.

check_config

- `--check-config`
- `False`

Check the configuration.

print_config

- `--print-config`
- `False`

Print the configuration settings as fully resolved. Implies *check_config*.

2.4.3 Logging

accesslog

- `--access-logfile FILE`
- `None`

The Access log file to write to.

`'-'` means log to stdout.

disable_redirect_access_to_syslog

- `--disable-redirect-access-to-syslog`
- `False`

Disable redirect access logs to syslog.

New in version 19.8.

access_log_format

- `--access-logformat STRING`
- `%(h)s %(l)s %(u)s %(t)s "%(r)s" %(s)s %(b)s "%(f)s" "%(a)s"`

The access log format.

Identifier	Description
h	remote address
l	`'-'`
u	user name
t	date of the request
r	status line (e.g. `GET / HTTP/1.1`)
m	request method
U	URL path without query string
q	query string
H	protocol
s	status
B	response length
b	response length or `'-'` (CLF format)
f	referer
a	user agent
T	request time in seconds
M	request time in milliseconds
D	request time in microseconds
L	request time in decimal seconds
p	process ID
{header}i	request header
{header}o	response header
{variable}e	environment variable

Use lowercase for header and environment variable names, and put `{...}x` names inside `%(...)s`. For example:

```
%({x-forwarded-for}i)s
```

errorlog

- `--error-logfile FILE, --log-file FILE`
- `-`

The Error log file to write to.

Using `'-'` for FILE makes gunicorn log to stderr.

Changed in version 19.2: Log to stderr by default.

loglevel

- `--log-level LEVEL`
- `info`

The granularity of Error log outputs.

Valid level names are:

- debug
- info
- warning
- error
- critical

capture_output

- `--capture-output`
- `False`

Redirect stdout/stderr to specified file in *errorlog*.

New in version 19.6.

logger_class

- `--logger-class STRING`
- `gunicorn.glogging.Logger`

The logger you want to use to log events in Gunicorn.

The default class (`gunicorn.glogging.Logger`) handle most of normal usages in logging. It provides error and access logging.

You can provide your own logger by giving Gunicorn a Python path to a subclass like `gunicorn.glogging.Logger`.

logconfig

- `--log-config FILE`
- `None`

The log config file to use. Gunicorn uses the standard Python logging module's Configuration file format.

logconfig_dict

- `--log-config-dict`
- `{}`

The log config dictionary to use, using the standard Python logging module's dictionary configuration format. This option takes precedence over the *logconfig* option, which uses the older file configuration format.

Format: https://docs.python.org/3/library/logging.config.html#logging.config.dictConfig

New in version 19.8.

syslog_addr

- `--log-syslog-to SYSLOG_ADDR`
- `udp://localhost:514`

Address to send syslog messages.

Address is a string of the form:

- `unix://PATH#TYPE` : for unix domain socket. `TYPE` can be `stream` for the stream driver or `dgram` for the dgram driver. `stream` is the default.
- `udp://HOST:PORT` : for UDP sockets
- `tcp://HOST:PORT` : for TCP sockets

syslog

- `--log-syslog`
- `False`

Send *Gunicorn* logs to syslog.

Changed in version 19.8: You can now disable sending access logs by using the *disable_redirect_access_to_syslog* setting.

syslog_prefix

- `--log-syslog-prefix SYSLOG_PREFIX`
- `None`

Makes Gunicorn use the parameter as program-name in the syslog entries.

All entries will be prefixed by `gunicorn.<prefix>`. By default the program name is the name of the process.

syslog_facility

- `--log-syslog-facility SYSLOG_FACILITY`
- `user`

Syslog facility name

enable_stdio_inheritance

- `-R, --enable-stdio-inheritance`
- `False`

Enable stdio inheritance.

Enable inheritance for stdio file descriptors in daemon mode.

Note: To disable the Python stdout buffering, you can to set the user environment variable `PYTHONUNBUFFERED` .

statsd_host

- `--statsd-host STATSD_ADDR`
- `None`

`host:port` of the statsd server to log to.

New in version 19.1.

dogstatsd_tags

- `--dogstatsd-tags DOGSTATSD_TAGS`
- `(empty string)`

A comma-delimited list of datadog statsd (dogstatsd) tags to append to statsd metrics.

New in version 20.

statsd_prefix

- `--statsd-prefix STATSD_PREFIX`
- `(empty string)`

Prefix to use when emitting statsd metrics (a trailing `.` is added, if not provided).

New in version 19.2.

2.4.4 Process Naming

proc_name

- `-n STRING, --name STRING`
- `None`

A base to use with setproctitle for process naming.

This affects things like `ps` and `top`. If you're going to be running more than one instance of Gunicorn you'll probably want to set a name to tell them apart. This requires that you install the setproctitle module.

If not set, the *default_proc_name* setting will be used.

default_proc_name

- `gunicorn`

Internal setting that is adjusted for each type of application.

2.4.5 SSL

keyfile

- `--keyfile FILE`
- `None`

SSL key file

certfile

- `--certfile FILE`
- `None`

SSL certificate file

ssl_version

- `--ssl-version`
- `_SSLMethod.PROTOCOL_TLS`

SSL version to use.

–ssl-version	Description
SSLv3	SSLv3 is not-secure and is strongly discouraged.
SSLv23	Alias for TLS. Deprecated in Python 3.6, use TLS.
TLS	Negotiate highest possible version between client/server. Can yield SSL. (Python 3.6+)
TLSv1	TLS 1.0
TLSv1_1	TLS 1.1 (Python 3.4+)
TLSv1_2	TLS 1.2 (Python 3.4+)
TLS_SERVER	Auto-negotiate the highest protocol version like TLS, but only support server-side SSLSocket connections. (Python 3.6+)

Changed in version 19.7: The default value has been changed from `ssl.PROTOCOL_TLSv1` to `ssl.PROTOCOL_SSLv23`.

Changed in version 20.0: This setting now accepts string names based on `ssl.PROTOCOL_` constants.

cert_reqs

- `--cert-reqs`
- `VerifyMode.CERT_NONE`

Whether client certificate is required (see stdlib ssl module's)

ca_certs

- `--ca-certs FILE`
- `None`

CA certificates file

suppress_ragged_eofs

- `--suppress-ragged-eofs`
- `True`

Suppress ragged EOFs (see stdlib ssl module's)

do_handshake_on_connect

- `--do-handshake-on-connect`
- `False`

Whether to perform SSL handshake on socket connect (see stdlib ssl module's)

ciphers

- `--ciphers`
- `None`

SSL Cipher suite to use, in the format of an OpenSSL cipher list.

By default we use the default cipher list from Python's `ssl` module, which contains ciphers considered strong at the time of each Python release.

As a recommended alternative, the Open Web App Security Project (OWASP) offers a vetted set of strong cipher strings rated A+ to C-. OWASP provides details on user-agent compatibility at each security level.

See the OpenSSL Cipher List Format Documentation for details on the format of an OpenSSL cipher list.

2.4.6 Security

limit_request_line

- `--limit-request-line INT`
- `4094`

The maximum size of HTTP request line in bytes.

This parameter is used to limit the allowed size of a client's HTTP request-line. Since the request-line consists of the HTTP method, URI, and protocol version, this directive places a restriction on the length of a request-URI allowed for a request on the server. A server needs this value to be large enough to hold any of its resource names, including any information that might be passed in the query part of a GET request. Value is a number from 0 (unlimited) to 8190.

This parameter can be used to prevent any DDOS attack.

limit_request_fields

- `--limit-request-fields INT`
- `100`

Limit the number of HTTP headers fields in a request.

This parameter is used to limit the number of headers in a request to prevent DDOS attack. Used with the *limit_request_field_size* it allows more safety. By default this value is 100 and can't be larger than 32768.

limit_request_field_size

- `--limit-request-field_size INT`
- `8190`

Limit the allowed size of an HTTP request header field.

Value is a positive number or 0. Setting it to 0 will allow unlimited header field sizes.

Warning: Setting this parameter to a very high or unlimited value can open up for DDOS attacks.

2.4.7 Server Hooks

on_starting

- ```
  def on_starting(server):
      pass
  ```

Called just before the master process is initialized.

The callable needs to accept a single instance variable for the Arbiter.

on_reload

- ```
  def on_reload(server):
      pass
  ```

Called to recycle workers during a reload via SIGHUP.

The callable needs to accept a single instance variable for the Arbiter.

when_ready

- ```
  def when_ready(server):
      pass
  ```

Called just after the server is started.

The callable needs to accept a single instance variable for the Arbiter.

pre_fork

- ```
  def pre_fork(server, worker):
      pass
  ```

Called just before a worker is forked.

The callable needs to accept two instance variables for the Arbiter and new Worker.

post_fork

- ```
  def post_fork(server, worker):
      pass
  ```

Called just after a worker has been forked.

The callable needs to accept two instance variables for the Arbiter and new Worker.

post_worker_init

- ```
  def post_worker_init(worker):
      pass
  ```

Called just after a worker has initialized the application.

The callable needs to accept one instance variable for the initialized Worker.

worker_int

- ```
  def worker_int(worker):
      pass
  ```

Called just after a worker exited on SIGINT or SIGQUIT.

The callable needs to accept one instance variable for the initialized Worker.

worker_abort

- ```
  def worker_abort(worker):
      pass
  ```

Called when a worker received the SIGABRT signal.

This call generally happens on timeout.

The callable needs to accept one instance variable for the initialized Worker.

pre_exec

- ```
  def pre_exec(server):
      pass
  ```

Called just before a new master process is forked.

The callable needs to accept a single instance variable for the Arbiter.

pre_request

- ```
  def pre_request(worker, req):
      worker.log.debug("%s %s" % (req.method, req.path))
  ```

Called just before a worker processes the request.

The callable needs to accept two instance variables for the Worker and the Request.

post_request

- ```
  def post_request(worker, req, environ, resp):
      pass
  ```

Called after a worker processes the request.

The callable needs to accept two instance variables for the Worker and the Request.

child_exit

- ```
  def child_exit(server, worker):
      pass
  ```

Called just after a worker has been exited, in the master process.

The callable needs to accept two instance variables for the Arbiter and the just-exited Worker.

New in version 19.7.

worker_exit

- ```
  def worker_exit(server, worker):
      pass
  ```

Called just after a worker has been exited, in the worker process.

The callable needs to accept two instance variables for the Arbiter and the just-exited Worker.
  ```

nworkers_changed

- ```
  def nworkers_changed(server, new_value, old_value):
      pass
  ```

Called just after *num_workers* has been changed.

The callable needs to accept an instance variable of the Arbiter and two integers of number of workers after and before change.

If the number of workers is set for the first time, *old_value* would be `None`.

on_exit

- ```
  def on_exit(server):
      pass
  ```

Called just before exiting Gunicorn.

The callable needs to accept a single instance variable for the Arbiter.

2.4.8 Server Mechanics

preload_app

- `--preload`
- `False`

Load application code before the worker processes are forked.

By preloading an application you can save some RAM resources as well as speed up server boot times. Although, if you defer application loading to each worker process, you can reload your application code easily by restarting workers.

sendfile

- `--no-sendfile`
- `None`

Disables the use of `sendfile()`.

If not set, the value of the `SENDFILE` environment variable is used to enable or disable its usage.

New in version 19.2.

Changed in version 19.4: Swapped `--sendfile` with `--no-sendfile` to actually allow disabling.

Changed in version 19.6: added support for the `SENDFILE` environment variable

reuse_port

- `--reuse-port`
- `False`

Set the `SO_REUSEPORT` flag on the listening socket.

New in version 19.8.

chdir

- `--chdir`
- `/home/docs/checkouts/readthedocs.org/user_builds/gunicorn-docs/checkouts/20.x/docs/source`

Chdir to specified directory before apps loading.

daemon

- `-D, --daemon`
- `False`

Daemonize the Gunicorn process.

Detaches the server from the controlling terminal and enters the background.

raw_env

- `-e ENV, --env ENV`
- `[]`

Set environment variable (key=value).

Pass variables to the execution environment. Ex.:

```
$ gunicorn -b 127.0.0.1:8000 --env FOO=1 test:app
```

and test for the foo variable environment in your application.

pidfile

- `-p FILE, --pid FILE`
- `None`

A filename to use for the PID file.

If not set, no PID file will be written.

worker_tmp_dir

- `--worker-tmp-dir DIR`
- `None`

A directory to use for the worker heartbeat temporary file.

If not set, the default temporary directory will be used.

Note: The current heartbeat system involves calling `os.fchmod` on temporary file handlers and may block a worker for arbitrary time if the directory is on a disk-backed filesystem.

See *How do I avoid Gunicorn excessively blocking in os.fchmod?* for more detailed information and a solution for avoiding this problem.

user

- `-u USER, --user USER`
- `1005`

Switch worker processes to run as this user.

A valid user id (as an integer) or the name of a user that can be retrieved with a call to `pwd.getpwnam(value)` or `None` to not change the worker process user.

group

- `-g GROUP, --group GROUP`
- `205`

Switch worker process to run as this group.

A valid group id (as an integer) or the name of a user that can be retrieved with a call to `pwd.getgrnam(value)` or `None` to not change the worker processes group.

umask

- `-m INT, --umask INT`
- `0`

A bit mask for the file mode on files written by Gunicorn.

Note that this affects unix socket permissions.

A valid value for the `os.umask(mode)` call or a string compatible with `int(value, 0)` (`0` means Python guesses the base, so values like `0`, `0xFF`, `0022` are valid for decimal, hex, and octal representations)

initgroups

- `--initgroups`
- `False`

If true, set the worker process's group access list with all of the groups of which the specified username is a member, plus the specified group id.

New in version 19.7.

tmp_upload_dir

- None

Directory to store temporary request data as they are read.

This may disappear in the near future.

This path should be writable by the process permissions set for Gunicorn workers. If not specified, Gunicorn will choose a system generated temporary directory.

secure_scheme_headers

- {'X-FORWARDED-PROTOCOL': 'ssl', 'X-FORWARDED-PROTO': 'https', 'X-FORWARDED-SSL': 'on'}

A dictionary containing headers and values that the front-end proxy uses to indicate HTTPS requests. These tell Gunicorn to set wsgi.url_scheme to https, so your application can tell that the request is secure.

The dictionary should map upper-case header names to exact string values. The value comparisons are case-sensitive, unlike the header names, so make sure they're exactly what your front-end proxy sends when handling HTTPS requests.

It is important that your front-end proxy configuration ensures that the headers defined here can not be passed directly from the client.

forwarded_allow_ips

- --forwarded-allow-ips STRING
- 127.0.0.1

Front-end's IPs from which allowed to handle set secure headers. (comma separate).

Set to * to disable checking of Front-end IPs (useful for setups where you don't know in advance the IP address of Front-end, but you still trust the environment).

By default, the value of the FORWARDED_ALLOW_IPS environment variable. If it is not defined, the default is "127.0.0.1".

pythonpath

- --pythonpath STRING
- None

A comma-separated list of directories to add to the Python path.

e.g. '/home/djangoprojects/myproject,/home/python/mylibrary'.

paste

- --paste STRING, --paster STRING
- None

Load a PasteDeploy config file. The argument may contain a # symbol followed by the name of an app section from the config file, e.g. `production.ini#admin`.

At this time, using alternate server blocks is not supported. Use the command line arguments to control server configuration instead.

proxy_protocol

- `--proxy-protocol`
- `False`

Enable detect PROXY protocol (PROXY mode).

Allow using HTTP and Proxy together. It may be useful for work with stunnel as HTTPS frontend and Gunicorn as HTTP server.

PROXY protocol: http://haproxy.1wt.eu/download/1.5/doc/proxy-protocol.txt

Example for stunnel config:

```
[https]
protocol = proxy
accept  = 443
connect = 80
cert = /etc/ssl/certs/stunnel.pem
key = /etc/ssl/certs/stunnel.key
```

proxy_allow_ips

- `--proxy-allow-from`
- `127.0.0.1`

Front-end's IPs from which allowed accept proxy requests (comma separate).

Set to * to disable checking of Front-end IPs (useful for setups where you don't know in advance the IP address of Front-end, but you still trust the environment)

raw_paste_global_conf

- `--paste-global CONF`
- `[]`

Set a PasteDeploy global config variable in `key=value` form.

The option can be specified multiple times.

The variables are passed to the the PasteDeploy entrypoint. Example:

```
$ gunicorn -b 127.0.0.1:8000 --paste development.ini --paste-global FOO=1 --paste-
↪global BAR=2
```

New in version 19.7.

strip_header_spaces

- `--strip-header-spaces`
- `False`

Strip spaces present between the header name and the the `:`.

This is known to induce vulnerabilities and is not compliant with the HTTP/1.1 standard. See https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn.

Use with care and only if necessary.

2.4.9 Server Socket

bind

- `-b ADDRESS, --bind ADDRESS`
- `['127.0.0.1:8000']`

The socket to bind.

A string of the form: `HOST`, `HOST:PORT`, `unix:PATH`, `fd://FD`. An IP is a valid `HOST`.

Changed in version 20.0: Support for `fd://FD` got added.

Multiple addresses can be bound. ex.:

```
$ gunicorn -b 127.0.0.1:8000 -b [::1]:8000 test:app
```

will bind the *test:app* application on localhost both on ipv6 and ipv4 interfaces.

If the `PORT` environment variable is defined, the default is `['0.0.0.0:$PORT']`. If it is not defined, the default is `['127.0.0.1:8000']`.

backlog

- `--backlog INT`
- `2048`

The maximum number of pending connections.

This refers to the number of clients that can be waiting to be served. Exceeding this number results in the client getting an error when attempting to connect. It should only affect servers under significant load.

Must be a positive integer. Generally set in the 64-2048 range.

2.4.10 Worker Processes

workers

- `-w INT, --workers INT`
- `1`

The number of worker processes for handling requests.

A positive integer generally in the `2-4 x $(NUM_CORES)` range. You'll want to vary this a bit to find the best for your particular application's work load.

By default, the value of the `WEB_CONCURRENCY` environment variable. If it is not defined, the default is `1`.

worker_class

- `-k STRING, --worker-class STRING`
- `sync`

The type of workers to use.

The default class (`sync`) should handle most "normal" types of workloads. You'll want to read *Design* for information on when you might want to choose one of the other worker classes. Required libraries may be installed using setuptools' `extras_require` feature.

A string referring to one of the following bundled classes:

- `sync`
- `eventlet` - Requires eventlet >= 0.24.1 (or install it via `pip install gunicorn[eventlet]`)
- `gevent` - Requires gevent >= 1.4 (or install it via `pip install gunicorn[gevent]`)
- `tornado` - Requires tornado >= 0.2 (or install it via `pip install gunicorn[tornado]`)
- `gthread` - Python 2 requires the futures package to be installed (or install it via `pip install gunicorn[gthread]`)

Optionally, you can provide your own worker by giving Gunicorn a Python path to a subclass of `gunicorn.workers.base.Worker`. This alternative syntax will load the gevent class: `gunicorn.workers.ggevent.GeventWorker`.

threads

- `--threads INT`
- `1`

The number of worker threads for handling requests.

Run each worker with the specified number of threads.

A positive integer generally in the `2-4 x $(NUM_CORES)` range. You'll want to vary this a bit to find the best for your particular application's work load.

If it is not defined, the default is `1`.

This setting only affects the Gthread worker type.

Note: If you try to use the `sync` worker type and set the `threads` setting to more than 1, the `gthread` worker type will be used instead.

---

worker_connections

- `--worker-connections INT`
- `1000`

The maximum number of simultaneous clients.

This setting only affects the Eventlet and Gevent worker types.

max_requests

- `--max-requests INT`
- `0`

The maximum number of requests a worker will process before restarting.

Any value greater than zero will limit the number of requests a worker will process before automatically restarting. This is a simple method to help limit the damage of memory leaks.

If this is set to zero (the default) then the automatic worker restarts are disabled.

max_requests_jitter

- `--max-requests-jitter INT`
- `0`

The maximum jitter to add to the *max_requests* setting.

The jitter causes the restart per worker to be randomized by `randint(0, max_requests_jitter)`. This is intended to stagger worker restarts to avoid all workers restarting at the same time.

New in version 19.2.

timeout

- `-t INT, --timeout INT`
- `30`

Workers silent for more than this many seconds are killed and restarted.

Value is a positive number or 0. Setting it to 0 has the effect of infinite timeouts by disabling timeouts for all workers entirely.

Generally, the default of thirty seconds should suffice. Only set this noticeably higher if you're sure of the repercussions for sync workers. For the non sync workers it just means that the worker process is still communicating and is not tied to the length of time required to handle a single request.

graceful_timeout

- `--graceful-timeout INT`
- `30`

Timeout for graceful workers restart.

After receiving a restart signal, workers have this much time to finish serving requests. Workers still alive after the timeout (starting from the receipt of the restart signal) are force killed.

keepalive

- `--keep-alive INT`
- `2`

The number of seconds to wait for requests on a Keep-Alive connection.

Generally set in the 1-5 seconds range for servers with direct connection to the client (e.g. when you don't have separate load balancer). When Gunicorn is deployed behind a load balancer, it often makes sense to set this to a higher value.

Note: `sync` worker does not support persistent connections and will ignore this option.

2.5 Instrumentation

New in version 19.1.

Gunicorn provides an optional instrumentation of the arbiter and workers using the statsD protocol over UDP. Thanks to the `gunicorn.instrument.statsd` module, Gunicorn becomes a statsD client. The use of UDP cleanly isolates Gunicorn from the receiving end of the statsD metrics so that instrumentation does not cause Gunicorn to be held up by a slow statsD consumer.

To use statsD, just tell Gunicorn where the statsD server is:

```
$ gunicorn --statsd-host=localhost:8125 --statsd-prefix=service.app ...
```

The `Statsd` logger overrides `gunicorn.glogging.Logger` to track all requests. The following metrics are generated:

- `gunicorn.requests`: request rate per second
- `gunicorn.request.duration`: histogram of request duration (in millisecond)
- `gunicorn.workers`: number of workers managed by the arbiter (gauge)
- `gunicorn.log.critical`: rate of critical log messages
- `gunicorn.log.error`: rate of error log messages
- `gunicorn.log.warning`: rate of warning log messages
- `gunicorn.log.exception`: rate of exceptional log messages

See the statsd-host setting for more information.

2.6 Deploying Gunicorn

We strongly recommend using Gunicorn behind a proxy server.

2.6.1 Nginx Configuration

Although there are many HTTP proxies available, we strongly advise that you use Nginx. If you choose another proxy server you need to make sure that it buffers slow clients when you use default Gunicorn workers. Without this buffering Gunicorn will be easily susceptible to denial-of-service attacks. You can use Hey to check if your proxy is behaving properly.

An example configuration file for fast clients with Nginx:

Listing 1: **nginx.conf**

```
worker_processes 1;

user nobody nogroup;
# 'user nobody nobody;' for systems with 'nobody' as a group instead
error_log  /var/log/nginx/error.log warn;
pid /var/run/nginx.pid;

events {
  worker_connections 1024; # increase if you have lots of clients
  accept_mutex off; # set to 'on' if nginx worker_processes > 1
  # 'use epoll;' to enable for Linux 2.6+
  # 'use kqueue;' to enable for FreeBSD, OSX
}

http {
  include mime.types;
  # fallback in case we can't determine a type
  default_type application/octet-stream;
  access_log /var/log/nginx/access.log combined;
  sendfile on;

  upstream app_server {
    # fail_timeout=0 means we always retry an upstream even if it failed
    # to return a good HTTP response

    # for UNIX domain socket setups
    server unix:/tmp/gunicorn.sock fail_timeout=0;

    # for a TCP configuration
    # server 192.168.0.7:8000 fail_timeout=0;
  }

  server {
    # if no Host match, close the connection to prevent host spoofing
    listen 80 default_server;
    return 444;
  }

  server {
    # use 'listen 80 deferred;' for Linux
    # use 'listen 80 accept_filter=httpready;' for FreeBSD
    listen 80;
    client_max_body_size 4G;

    # set the correct host(s) for your site
    server_name example.com www.example.com;
```

(continues on next page)

(continued from previous page)

```
    keepalive_timeout 5;

    # path for static files
    root /path/to/app/current/public;

    location / {
      # checks for static file, if not found proxy to app
      try_files $uri @proxy_to_app;
    }

    location @proxy_to_app {
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-Forwarded-Proto $scheme;
      proxy_set_header Host $http_host;
      # we don't want nginx trying to do something clever with
      # redirects, we set the Host: header above already.
      proxy_redirect off;
      proxy_pass http://app_server;
    }

    error_page 500 502 503 504 /500.html;
    location = /500.html {
      root /path/to/app/current/public;
    }
  }
}
```

If you want to be able to handle streaming request/responses or other fancy features like Comet, Long polling, or Web sockets, you need to turn off the proxy buffering. **When you do this** you must run with one of the async worker classes.

To turn off buffering, you only need to add `proxy_buffering off;` to your `location` block:

```
...
location @proxy_to_app {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_redirect off;
    proxy_buffering off;

    proxy_pass http://app_server;
}
...
```

It is recommended to pass protocol information to Gunicorn. Many web frameworks use this information to generate URLs. Without this information, the application may mistakenly generate 'http' URLs in 'https' responses, leading to mixed content warnings or broken applications. To configure Nginx to pass an appropriate header, add a `proxy_set_header` directive to your `location` block:

```
...
proxy_set_header X-Forwarded-Proto $scheme;
...
```

If you are running Nginx on a different host than Gunicorn you need to tell Gunicorn to trust the `X-Forwarded-*` headers sent by Nginx. By default, Gunicorn will only trust these headers if the connection comes from localhost. This is to prevent a malicious client from forging these headers:

```
$ gunicorn -w 3 --forwarded-allow-ips="10.170.3.217,10.170.3.220" test:app
```

When the Gunicorn host is completely firewalled from the external network such that all connections come from a trusted proxy (e.g. Heroku) this value can be set to '*'. Using this value is **potentially dangerous** if connections to Gunicorn may come from untrusted proxies or directly from clients since the application may be tricked into serving SSL-only content over an insecure connection.

Gunicorn 19 introduced a breaking change concerning how `REMOTE_ADDR` is handled. Previous to Gunicorn 19 this was set to the value of `X-Forwarded-For` if received from a trusted proxy. However, this was not in compliance with **RFC 3875** which is why the `REMOTE_ADDR` is now the IP address of **the proxy** and **not the actual user**.

To have access logs indicate **the actual user** IP when proxied, set *access_log_format* with a format which includes `X-Forwarded-For`. For example, this format uses `X-Forwarded-For` in place of `REMOTE_ADDR`:

```
%({x-forwarded-for}i)s %(l)s %(u)s %(t)s "%(r)s" %(s)s %(b)s "%(f)s" "%(a)s"
```

It is also worth noting that the `REMOTE_ADDR` will be completely empty if you bind Gunicorn to a UNIX socket and not a TCP `host:port` tuple.

2.6.2 Using Virtualenv

To serve an app from a Virtualenv it is generally easiest to just install Gunicorn directly into the Virtualenv. This will create a set of Gunicorn scripts for that Virtualenv which can be used to run applications normally.

If you have Virtualenv installed, you should be able to do something like this:

```
$ mkdir ~/venvs/
$ virtualenv ~/venvs/webapp
$ source ~/venvs/webapp/bin/activate
$ pip install gunicorn
$ deactivate
```

Then you just need to use one of the three Gunicorn scripts that was installed into `~/venvs/webapp/bin`.

Note: You can force the installation of Gunicorn in your Virtualenv by passing `-I` or `--ignore-installed` option to pip:

```
$ source ~/venvs/webapp/bin/activate
$ pip install -I gunicorn
```

2.6.3 Monitoring

Note: Make sure that when using either of these service monitors you do not enable the Gunicorn's daemon mode. These monitors expect that the process they launch will be the process they need to monitor. Daemonizing will fork-exec which creates an unmonitored process and generally just confuses the monitor services.

Gaffer

Using Gafferd and gaffer

Gaffer can be used to monitor Gunicorn. A simple configuration is:

```
[process:gunicorn]
cmd = gunicorn -w 3 test:app
cwd = /path/to/project
```

Then you can easily manage Gunicorn using Gaffer.

Using a Procfile

Create a `Procfile` in your project:

```
gunicorn = gunicorn -w 3 test:app
```

You can launch any other applications that should be launched at the same time.

Then you can start your Gunicorn application using Gaffer:

```
gaffer start
```

If gafferd is launched you can also load your Procfile in it directly:

```
gaffer load
```

All your applications will be then supervised by gafferd.

Runit

A popular method for deploying Gunicorn is to have it monitored by runit. Here is an example service definition:

```
#!/bin/sh

GUNICORN=/usr/local/bin/gunicorn
ROOT=/path/to/project
PID=/var/run/gunicorn.pid

APP=main:application

if [ -f $PID ]; then rm $PID; fi

cd $ROOT
exec $GUNICORN -c $ROOT/gunicorn.conf.py --pid=$PID $APP
```

Save this as `/etc/sv/[app_name]/run`, and make it executable (`chmod u+x /etc/sv/[app_name]/run`). Then run `ln -s /etc/sv/[app_name] /etc/service/[app_name]`. If runit is installed, Gunicorn should start running automatically as soon as you create the symlink.

If it doesn't start automatically, run the script directly to troubleshoot.

Supervisor

Another useful tool to monitor and control Gunicorn is Supervisor. A simple configuration is:

```
[program:gunicorn]
command=/path/to/gunicorn main:application -c /path/to/gunicorn.conf.py
directory=/path/to/project
```

(continues on next page)

(continued from previous page)

```
user=nobody
autostart=true
autorestart=true
redirect_stderr=true
```

Upstart

Using Gunicorn with upstart is simple. In this example we will run the app "myapp" from a virtualenv. All errors will go to `/var/log/upstart/myapp.log`.

/etc/init/myapp.conf:

```
description "myapp"

start on (filesystem)
stop on runlevel [016]

respawn
setuid nobody
setgid nogroup
chdir /path/to/app/directory

exec /path/to/virtualenv/bin/gunicorn myapp:app
```

Systemd

A tool that is starting to be common on linux systems is Systemd. It is a system services manager that allows for strict process management, resources and permissions control.

Below are configurations files and instructions for using systemd to create a unix socket for incoming Gunicorn requests. Systemd will listen on this socket and start gunicorn automatically in response to traffic. Later in this section are instructions for configuring Nginx to forward web traffic to the newly created unix socket:

/etc/systemd/system/gunicorn.service:

```
[Unit]
Description=gunicorn daemon
Requires=gunicorn.socket
After=network.target

[Service]
Type=notify
# the specific user that our service will run as
User=someuser
Group=someuser
# another option for an even more restricted service is
# DynamicUser=yes
# see http://0pointer.net/blog/dynamic-users-with-systemd.html
RuntimeDirectory=gunicorn
WorkingDirectory=/home/someuser/applicationroot
ExecStart=/usr/bin/gunicorn applicationname.wsgi
ExecReload=/bin/kill -s HUP $MAINPID
KillMode=mixed
TimeoutStopSec=5
```

(continues on next page)

(continued from previous page)

```
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

/etc/systemd/system/gunicorn.socket:

```
[Unit]
Description=gunicorn socket

[Socket]
ListenStream=/run/gunicorn.sock
# Our service won't need permissions for the socket, since it
# inherits the file descriptor by socket activation
# only the nginx daemon will need access to the socket
SocketUser=www-data
# Optionally restrict the socket permissions even more.
# SocketMode=600

[Install]
WantedBy=sockets.target
```

Next enable and start the socket (it will autostart at boot too):

```
systemctl enable --now gunicorn.socket
```

Now let's see if the nginx daemon will be able to connect to the socket. Running `sudo -u www-data curl --unix-socket /run/gunicorn.sock http`, our Gunicorn service will be automatically started and you should see some HTML from your server in the terminal.

Note: systemd employs cgroups to track the processes of a service, so it doesn't need pid files. In the rare case that you need to find out the service main pid, you can use `systemctl show --value -p MainPID gunicorn.service`, but if you only want to send a signal an even better option is `systemctl kill -s HUP gunicorn.service`.

Note: `www-data` is the default nginx user in debian, other distributions use different users (for example: `http` or `nginx`). Check your distro to know what to put for the socket user, and for the sudo command.

You must now configure your web proxy to send traffic to the new Gunicorn socket. Edit your `nginx.conf` to include the following:

/etc/nginx/nginx.conf:

```
user www-data;
...
http {
    server {
        listen          8000;
        server_name     127.0.0.1;
        location / {
            proxy_pass http://unix:/run/gunicorn.sock;
        }
    }
```

(continues on next page)

(continued from previous page)

```
}
...
```

Note: The listen and server_name used here are configured for a local machine. In a production server you will most likely listen on port 80, and use your URL as the server_name.

Now make sure you enable the nginx service so it automatically starts at boot:

```
systemctl enable nginx.service
```

Either reboot, or start Nginx with the following command:

```
systemctl start nginx
```

Now you should be able to test Nginx with Gunicorn by visiting http://127.0.0.1:8000/ in any web browser. Systemd is now set up.

2.6.4 Logging

Logging can be configured by using various flags detailed in the configuration documentation or by creating a logging configuration file. Send the `USR1` signal to rotate logs if you are using the logrotate utility:

```
kill -USR1 $(cat /var/run/gunicorn.pid)
```

Note: Overriding the `LOGGING` dictionary requires to set `disable_existing_loggers: False` to not interfere with the Gunicorn logging.

Warning: Gunicorn error log is here to log errors from Gunicorn, not from another application.

2.7 Signal Handling

A brief description of the signals handled by Gunicorn. We also document the signals used internally by Gunicorn to communicate with the workers.

2.7.1 Master process

- `QUIT`, `INT`: Quick shutdown
- `TERM`: Graceful shutdown. Waits for workers to finish their current requests up to the *graceful_timeout*.
- `HUP`: Reload the configuration, start the new worker processes with a new configuration and gracefully shutdown older workers. If the application is not preloaded (using the *preload_app* option), Gunicorn will also load the new version of it.
- `TTIN`: Increment the number of processes by one
- `TTOU`: Decrement the number of processes by one

- USR1: Reopen the log files
- USR2: Upgrade Gunicorn on the fly. A separate TERM signal should be used to kill the old master process. This signal can also be used to use the new versions of pre-loaded applications. See *Upgrading to a new binary on the fly* for more information.
- WINCH: Gracefully shutdown the worker processes when Gunicorn is daemonized.

2.7.2 Worker process

Sending signals directly to the worker processes should not normally be needed. If the master process is running, any exited worker will be automatically respawned.

- QUIT, INT: Quick shutdown
- TERM: Graceful shutdown
- USR1: Reopen the log files

2.7.3 Reload the configuration

The HUP signal can be used to reload the Gunicorn configuration on the fly.

```
2013-06-29 06:26:55 [20682] [INFO] Handling signal: hup
2013-06-29 06:26:55 [20682] [INFO] Hang up: Master
2013-06-29 06:26:55 [20703] [INFO] Booting worker with pid: 20703
2013-06-29 06:26:55 [20702] [INFO] Booting worker with pid: 20702
2013-06-29 06:26:55 [20688] [INFO] Worker exiting (pid: 20688)
2013-06-29 06:26:55 [20687] [INFO] Worker exiting (pid: 20687)
2013-06-29 06:26:55 [20689] [INFO] Worker exiting (pid: 20689)
2013-06-29 06:26:55 [20704] [INFO] Booting worker with pid: 20704
```

Sending a HUP signal will reload the configuration, start the new worker processes with a new configuration and gracefully shutdown older workers. If the application is not preloaded (using the *preload_app* option), Gunicorn will also load the new version of it.

2.7.4 Upgrading to a new binary on the fly

Changed in version 19.6.0: PID file naming format has been changed from <name>.pid.oldbin to <name>.pid.2.

If you need to replace the Gunicorn binary with a new one (when upgrading to a new version or adding/removing server modules), you can do it without any service downtime - no incoming requests will be lost. Preloaded applications will also be reloaded.

First, replace the old binary with a new one, then send a USR2 signal to the current master process. It executes a new binary whose PID file is postfixed with .2 (e.g. /var/run/gunicorn.pid.2), which in turn starts a new master process and new worker processes:

```
  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
20844 benoitc   20   0 54808  11m 3352 S   0.0  0.1   0:00.36 gunicorn: master␣
↪[test:app]
20849 benoitc   20   0 54808 9.9m 1500 S   0.0  0.1   0:00.02 gunicorn: worker␣
↪[test:app]
20850 benoitc   20   0 54808 9.9m 1500 S   0.0  0.1   0:00.01 gunicorn: worker␣
↪[test:app]
```

(continues on next page)

(continued from previous page)

```
20851 benoitc   20   0 54808 9.9m 1500 S   0.0  0.1   0:00.01 gunicorn: worker␣
↪[test:app]
20854 benoitc   20   0 55748  12m 3348 S   0.0  0.2   0:00.35 gunicorn: master␣
↪[test:app]
20859 benoitc   20   0 55748  11m 1500 S   0.0  0.1   0:00.01 gunicorn: worker␣
↪[test:app]
20860 benoitc   20   0 55748  11m 1500 S   0.0  0.1   0:00.00 gunicorn: worker␣
↪[test:app]
20861 benoitc   20   0 55748  11m 1500 S   0.0  0.1   0:00.01 gunicorn: worker␣
↪[test:app]
```

At this point, two instances of Gunicorn are running, handling the incoming requests together. To phase the old instance out, you have to send a `WINCH` signal to the old master process, and its worker processes will start to gracefully shut down.

At this point you can still revert to the old process since it hasn't closed its listen sockets yet, by following these steps:

- Send a `HUP` signal to the old master process - it will start the worker processes without reloading a configuration file
- Send a `TERM` signal to the new master process to gracefully shut down its worker processes
- Send a `QUIT` signal to the new master process to force it quit

If for some reason the new worker processes do not quit, send a `KILL` signal to them after the new master process quits, and everything will back to exactly as before the upgrade attempt.

If the update is successful and you want to keep the new master process, send a `TERM` signal to the old master process to leave only the new server running:

```
  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
20854 benoitc   20   0 55748  12m 3348 S   0.0  0.2   0:00.45 gunicorn: master␣
↪[test:app]
20859 benoitc   20   0 55748  11m 1500 S   0.0  0.1   0:00.02 gunicorn: worker␣
↪[test:app]
20860 benoitc   20   0 55748  11m 1500 S   0.0  0.1   0:00.02 gunicorn: worker␣
↪[test:app]
20861 benoitc   20   0 55748  11m 1500 S   0.0  0.1   0:00.01 gunicorn: worker␣
↪[test:app]
```

2.8 Custom Application

New in version 19.0.

Sometimes, you want to integrate Gunicorn with your WSGI application. In this case, you can inherit from `gunicorn.app.base.BaseApplication`.

Here is a small example where we create a very small WSGI app and load it with a custom Application:

```
import multiprocessing

import gunicorn.app.base


def number_of_workers():
    return (multiprocessing.cpu_count() * 2) + 1
```

(continues on next page)

(continued from previous page)

```
def handler_app(environ, start_response):
    response_body = b'Works fine'
    status = '200 OK'

    response_headers = [
        ('Content-Type', 'text/plain'),
    ]

    start_response(status, response_headers)

    return [response_body]


class StandaloneApplication(gunicorn.app.base.BaseApplication):

    def __init__(self, app, options=None):
        self.options = options or {}
        self.application = app
        super().__init__()

    def load_config(self):
        config = {key: value for key, value in self.options.items()
                  if key in self.cfg.settings and value is not None}
        for key, value in config.items():
            self.cfg.set(key.lower(), value)

    def load(self):
        return self.application


if __name__ == '__main__':
    options = {
        'bind': '%s:%s' % ('127.0.0.1', '8080'),
        'workers': number_of_workers(),
    }
    StandaloneApplication(handler_app, options).run()
```

2.8.1 Direct Usage of Existing WSGI Apps

If necessary, you can run Gunicorn straight from Python, allowing you to specify a WSGI-compatible application at runtime. This can be handy for rolling deploys or in the case of using PEX files to deploy your application, as the app and Gunicorn can be bundled in the same PEX file. Gunicorn has this functionality built-in as a first class citizen known as `gunicorn.app.wsgiapp`. This can be used to run WSGI-compatible app instances such as those produced by Flask or Django. Assuming your WSGI API package is *exampleapi*, and your application instance is *app*, this is all you need to get going:

```
gunicorn.app.wsgiapp exampleapi:app
```

This command will work with any Gunicorn CLI parameters or a config file - just pass them along as if you're directly giving them to Gunicorn:

```
# Custom parameters
$ python gunicorn.app.wsgiapp exampleapi:app --bind=0.0.0.0:8081 --workers=4
# Using a config file
$ python gunicorn.app.wsgiapp exampleapi:app -c config.py
```

Note for those using PEX: use `-c gunicorn` as your entry at build time, and your compiled app should work with the entry point passed to it at run time.

```
# Generic pex build command via bash from root of exampleapi project
$ pex . -v -c gunicorn -o compiledapp.pex
# Running it
./compiledapp.pex exampleapi:app -c gunicorn_config.py
```

2.9 Design

A brief description of the architecture of Gunicorn.

2.9.1 Server Model

Gunicorn is based on the pre-fork worker model. This means that there is a central master process that manages a set of worker processes. The master never knows anything about individual clients. All requests and responses are handled completely by worker processes.

Master

The master process is a simple loop that listens for various process signals and reacts accordingly. It manages the list of running workers by listening for signals like TTIN, TTOU, and CHLD. TTIN and TTOU tell the master to increase or decrease the number of running workers. CHLD indicates that a child process has terminated, in this case the master process automatically restarts the failed worker.

Sync Workers

The most basic and the default worker type is a synchronous worker class that handles a single request at a time. This model is the simplest to reason about as any errors will affect at most a single request. Though as we describe below only processing a single request at a time requires some assumptions about how applications are programmed.

`sync` worker does not support persistent connections - each connection is closed after response has been sent (even if you manually add `Keep-Alive` or `Connection: keep-alive` header in your application).

Async Workers

The asynchronous workers available are based on Greenlets (via Eventlet and Gevent). Greenlets are an implementation of cooperative multi-threading for Python. In general, an application should be able to make use of these worker classes with no changes.

For full greenlet support applications might need to be adapted. When using, e.g., Gevent and Psycopg it makes sense to ensure psycogreen is installed and setup.

Other applications might not be compatible at all as they, e.g., rely on the original unpatched behavior.

Tornado Workers

There's also a Tornado worker class. It can be used to write applications using the Tornado framework. Although the Tornado workers are capable of serving a WSGI application, this is not a recommended configuration.

AsyncIO Workers

These workers are compatible with Python 3.

The worker *gthread* is a threaded worker. It accepts connections in the main loop, accepted connections are added to the thread pool as a connection job. On keepalive connections are put back in the loop waiting for an event. If no event happen after the keep alive timeout, the connection is closed.

You can port also your application to use aiohttp's *web.Application*' API and use the `aiohttp.worker.GunicornWebWorker` worker.

2.9.2 Choosing a Worker Type

The default synchronous workers assume that your application is resource-bound in terms of CPU and network bandwidth. Generally this means that your application shouldn't do anything that takes an undefined amount of time. An example of something that takes an undefined amount of time is a request to the internet. At some point the external network will fail in such a way that clients will pile up on your servers. So, in this sense, any web application which makes outgoing requests to APIs will benefit from an asynchronous worker.

This resource bound assumption is why we require a buffering proxy in front of a default configuration Gunicorn. If you exposed synchronous workers to the internet, a DOS attack would be trivial by creating a load that trickles data to the servers. For the curious, Hey is an example of this type of load.

Some examples of behavior requiring asynchronous workers:

- Applications making long blocking calls (Ie, external web services)
- Serving requests directly to the internet
- Streaming requests and responses
- Long polling
- Web sockets
- Comet

2.9.3 How Many Workers?

DO NOT scale the number of workers to the number of clients you expect to have. Gunicorn should only need 4-12 worker processes to handle hundreds or thousands of requests per second.

Gunicorn relies on the operating system to provide all of the load balancing when handling requests. Generally we recommend `(2 x $num_cores) + 1` as the number of workers to start off with. While not overly scientific, the formula is based on the assumption that for a given core, one worker will be reading or writing from the socket while the other worker is processing a request.

Obviously, your particular hardware and application are going to affect the optimal number of workers. Our recommendation is to start with the above guess and tune using TTIN and TTOU signals while the application is under load.

Always remember, there is such a thing as too many workers. After a point your worker processes will start thrashing system resources decreasing the throughput of the entire system.

2.9.4 How Many Threads?

Since Gunicorn 19, a threads option can be used to process requests in multiple threads. Using threads assumes use of the gthread worker. One benefit from threads is that requests can take longer than the worker timeout while notifying the master process that it is not frozen and should not be killed. Depending on the system, using multiple threads, multiple worker processes, or some mixture, may yield the best results. For example, CPython may not perform as well as Jython when using threads, as threading is implemented differently by each. Using threads instead of processes is a good way to reduce the memory footprint of Gunicorn, while still allowing for application upgrades using the reload signal, as the application code will be shared among workers but loaded only in the worker processes (unlike when using the preload setting, which loads the code in the master process).

Note: Under Python 2.x, you need to install the 'futures' package to use this feature.

2.10 FAQ

2.10.1 WSGI Bits

How do I set SCRIPT_NAME?

By default `SCRIPT_NAME` is an empty string. The value could be set by setting `SCRIPT_NAME` in the environment or as an HTTP header.

2.10.2 Server Stuff

How do I reload my application in Gunicorn?

You can gracefully reload by sending HUP signal to gunicorn:

```
$ kill -HUP masterpid
```

How might I test a proxy configuration?

The Hey program is a great way to test that your proxy is correctly buffering responses for the synchronous workers:

```
$ hey -n 10000 -c 100 http://127.0.0.1:5000/
```

This runs a benchmark of 10000 requests with 100 running concurrently.

How can I name processes?

If you install the Python package setproctitle Gunicorn will set the process names to something a bit more meaningful. This will affect the output you see in tools like `ps` and `top`. This helps for distinguishing the master process as well as between masters when running more than one app on a single machine. See the proc_name setting for more information.

Why is there no HTTP Keep-Alive?

The default Sync workers are designed to run behind Nginx which only uses HTTP/1.0 with its upstream servers. If you want to deploy Gunicorn to handle unbuffered requests (ie, serving requests directly from the internet) you should use one of the async workers.

2.10.3 Worker Processes

How do I know which type of worker to use?

Read the *Design* page for help on the various worker types.

What types of workers are there?

Check out the configuration docs for worker_class.

How can I figure out the best number of worker processes?

Here is our recommendation for tuning the number of workers.

How can I change the number of workers dynamically?

TTIN and TTOU signals can be sent to the master to increase or decrease the number of workers.

To increase the worker count by one:

```
$ kill -TTIN $masterpid
```

To decrease the worker count by one:

```
$ kill -TTOU $masterpid
```

Does Gunicorn suffer from the thundering herd problem?

The thundering herd problem occurs when many sleeping request handlers, which may be either threads or processes, wake up at the same time to handle a new request. Since only one handler will receive the request, the others will have been awakened for no reason, wasting CPU cycles. At this time, Gunicorn does not implement any IPC solution for coordinating between worker processes. You may experience high load due to this problem when using many workers or threads. However a work has been started to remove this issue.

Why I don't see any logs in the console?

In version 19.0, Gunicorn doesn't log by default in the console. To watch the logs in the console you need to use the option `--log-file=-`. In version 19.2, Gunicorn logs to the console by default again.

2.10.4 Kernel Parameters

When dealing with large numbers of concurrent connections there are a handful of kernel parameters that you might need to adjust. Generally these should only affect sites with a very large concurrent load. These parameters are not specific to Gunicorn, they would apply to any sort of network server you may be running.

These commands are for Linux. Your particular OS may have slightly different parameters.

How can I increase the maximum number of file descriptors?

One of the first settings that usually needs to be bumped is the maximum number of open file descriptors for a given process. For the confused out there, remember that Unices treat sockets as files.

```
$ sudo ulimit -n 2048
```

How can I increase the maximum socket backlog?

Listening sockets have an associated queue of incoming connections that are waiting to be accepted. If you happen to have a stampede of clients that fill up this queue new connections will eventually start getting dropped.

```
$ sudo sysctl -w net.core.somaxconn="2048"
```

How can I disable the use of `sendfile()`

Disabling the use `sendfile()` can be done by using the `--no-sendfile` setting or by setting the environment variable `SENDFILE` to 0.

2.10.5 Troubleshooting

How do I fix Django reporting an `ImproperlyConfigured` error?

With asynchronous workers, creating URLs with the `reverse` function of `django.core.urlresolvers` may fail. Use `reverse_lazy` instead.

How do I avoid Gunicorn excessively blocking in `os.fchmod`?

The current heartbeat system involves calling `os.fchmod` on temporary file handlers and may block a worker for arbitrary time if the directory is on a disk-backed filesystem. For example, by default `/tmp` is not mounted as `tmpfs` in Ubuntu; in AWS an EBS root instance volume may sometimes hang for half a minute and during this time Gunicorn workers may completely block in `os.fchmod`. `os.fchmod` may introduce extra delays if the disk gets full. Also Gunicorn may refuse to start if it can't create the files when the disk is full.

Currently to avoid these problems you can use a `tmpfs` mount (for a new directory or for `/tmp`) and pass its path to `--worker-tmp-dir`. First, check whether your `/tmp` is disk-backed or RAM-backed:

```
$ df /tmp
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/xvda1           ...     ...       ...  ... /
```

No luck. If you are using Fedora or Ubuntu, you should already have a `tmpfs` mount at `/dev/shm`:

```
$ df /dev/shm
Filesystem     1K-blocks     Used Available Use% Mounted on
tmpfs                ...      ...       ...  ... /dev/shm
```

In this case you can set `--worker-tmp-dir /dev/shm`, otherwise you can create a new `tmpfs` mount:

```
sudo cp /etc/fstab /etc/fstab.orig
sudo mkdir /mem
echo 'tmpfs       /mem tmpfs defaults,size=64m,mode=1777,noatime,comment=for-gunicorn␣
↪0 0' | sudo tee -a /etc/fstab
sudo mount /mem
```

Check the result:

```
$ df /mem
Filesystem     1K-blocks  Used Available Use% Mounted on
tmpfs              65536     0     65536   0% /mem
```

Now you can set `--worker-tmp-dir /mem`.

Why are Workers Silently Killed?

A sometimes subtle problem to debug is when a worker process is killed and there is little logging information about what happened.

If you use a reverse proxy like NGINX you might see 502 returned to a client.

In the gunicorn logs you might simply see `[35] [INFO] Booting worker with pid:  35`

It's completely normal for workers to be killed and startup, for example due to max-requests setting. Ordinarily gunicorn will capture any signals and log something.

This particular failure case is usually due to a SIGKILL being received, as it's not possible to catch this signal silence is usually a common side effect! A common cause of SIGKILL is when OOM killer terminates a process due to low memory condition.

This is increasingly common in container deployments where memory limits are enforced by cgroups, you'll usually see evidence of this from dmesg:

```
dmesg | grep gunicorn
Memory cgroup out of memory: Kill process 24534 (gunicorn) score 1506 or sacrifice␣
↪child
Killed process 24534 (gunicorn) total-vm:1016648kB, anon-rss:550160kB, file-
↪rss:25824kB, shmem-rss:0kB
```

In these instances adjusting the memory limit is usually your best bet, it's also possible to configure OOM not to send SIGKILL by default.

2.11 Community

Use these channels to communicate about the project.

2.11.1 Project Management & Discussions

Gunicorn uses GitHub for the project management. GitHub issues are used for 3 different purposes:

- Bug tracker : to check latest bug
- Forum : Stackoverflow-style questions about Gunicorn usage
- Mailing list : Discussion of Gunicorn development, new features and project management.

Project maintenance guidelines are avaible on the wiki .

2.11.2 IRC

The Gunicorn channel is on the Freenode IRC network. You can chat with other on #gunicorn channel.

2.11.3 Issue Tracking

Bug reports, enhancement requests and tasks generally go in the Github issue tracker.

2.11.4 Security Issues

The security mailing list is a place to report security issues. Only developers are subscribed to it. To post a message to the list use the address to security@gunicorn.org .

2.12 Changelog

2.12.1 20.0.4 / 2019/11/26

- fix binding a socket using the file descriptor
- remove support for the *bdist_rpm* build

2.12.2 20.0.3 / 2019/11/24

- fixed load of a config file without a Python extension
- fixed *socketfromfd.fromfd* when defaults are not set

Note: we now warn when we load a config file without Python Extension

---

2.12.3 20.0.2 / 2019/11/23

- fix changelog

2.12.4 20.0.1 / 2019/11/23

- fixed the way the config module is loaded. *__file__* is now available
- fixed *wsgi.input_terminated*. It is always true.
- use the highest protocol version of openssl by default

- only support Python >= 3.5
- added *__repr__* method to *Config* instance
- fixed support of AIX platform and musl libc in *socketfromfd.fromfd* function
- fixed support of applications loaded from a factory function
- fixed chunked encoding support to prevent any request smuggling
- Capture os.sendfile before patching in gevent and eventlet workers. fix *RecursionError*.
- removed locking in reloader when adding new files
- load the WSGI application before the loader to pick up all files

---

Note: this release add official support for applications loaded from a factory function as documented in Flask and other places.

2.12.5 19.10.0 / 2019/11/23

- unblock select loop during reload of a sync worker
- security fix: http desync attack
- handle *wsgi.input_terminated*
- added support for str and bytes in unix socket addresses
- fixed *max_requests* setting
- headers values are now encoded as LATN1, not ASCII
- fixed *InotifyReloadeder*: handle *module.__file__* is None
- fixed compatibility with tornado 6
- fixed root logging
- Prevent removalof unix sockets from *reuse_port*
- Clear tornado ioloop before os.fork
- Miscellaneous fixes and improvement for linting using Pylint

2.12.6 20.0 / 2019/10/30

- Fixed *fdopen RuntimeWarning* in Python 3.8
- Added check and exception for str type on value in Response process_headers method.
- Ensure WSGI header value is string before conducting regex search on it.
- Added pypy3 to list of tested environments
- Grouped *StopIteration* and *KeyboardInterrupt* exceptions with same body together in Arbiter.run()
- Added *setproctitle* module to *extras_require* in setup.py
- Avoid unnecessary chown of temporary files
- Logging: Handle auth type case insensitively
- Removed *util.import_module*

- Removed fallback for *types.SimpleNamespace* in tests utils
- Use *SourceFileLoader* instead instead of *execfile_*
- Use *importlib* instead of *__import__* and eval‘
- Fixed eventlet patching
- Added optional datadog tags for statsd metrics
- Header values now are encoded using latin-1, not ascii.
- Rewritten *parse_address* util added test
- Removed redundant super() arguments
- Simplify *futures* import in gthread module
- Fixed worker_connections‘ setting to also affects the Gthread worker type
- Fixed setting max_requests
- Bump minimum Eventlet and Gevent versions to 0.24 and 1.4
- Use Python default SSL cipher list by default
- handle *wsgi.input_terminated* extension
- Simplify Paste Deployment documentation
- Fix root logging: root and logger are same level.
- Fixed typo in ssl_version documentation
- Documented systemd deployement unit examples
- Added systemd sd_notify support
- Fixed typo in gthread.py
- Added tornado 5 and 6 support
- Declare our setuptools dependency
- Added support to *–bind* to open file descriptors
- Document how to serve WSGI app modules from Gunicorn
- Provide guidance on X-Forwarded-For access log in documentation
- Add support for named constants in the *–ssl-version* flag
- Clarify log format usage of header & environment in documentation
- Fixed systemd documentation to properly setup gunicorn unix socket
- Prevent removal unix socket for reuse_port
- Fix *ResourceWarning* when reading a Python config module
- Remove unnecessary call to dict keys method
- Support str and bytes for UNIX socket addresses
- fixed *InotifyReloadeder*: handle *module.__file__* is None
- */dev/shm* as a convenient alternative to making your own tmpfs mount in fchmod FAQ
- fix examples to work on python3
- Fix typo in *–max-requests* documentation

- Clear tornado ioloop before os.fork
- Miscellaneous fixes and improvement for linting using Pylint

Breaking Change

- Removed gaiohttp worker
- Drop support for Python 2.x
- Drop support for EOL Python 3.2 and 3.3
- Drop support for Paste Deploy server blocks

2.12.7 History

Changelog - 2018

Note: Please see *Changelog* for the latest changes

19.9.0 / 2018/07/03

- fix: address a regression that prevented syslog support from working (issue 1668, pull request 1773)
- fix: correctly set *REMOTE_ADDR* on versions of Python 3 affected by Python Issue 30205 (issue 1755, pull request 1796)
- fix: show zero response length correctly in access log (pull request 1787)
- fix: prevent raising `AttributeError` when `--reload` is not passed in case of a `SyntaxError` raised from the WSGI application. (issue 1805, pull request 1806)
- The internal module `gunicorn.workers.async` was renamed to `gunicorn.workers.base_async` since `async` is now a reserved word in Python 3.7. (pull request 1527)

19.8.1 / 2018/04/30

- fix: secure scheme headers when bound to a unix socket (issue 1766, pull request 1767)

19.8.0 / 2018/04/28

- Eventlet 0.21.0 support (issue 1584)
- Tornado 5 support (issue 1728, pull request 1752)
- support watching additional files with `--reload-extra-file` (pull request 1527)
- support configuring logging with a dictionary with `--logging-config-dict` (issue 1087, pull request 1110, pull request 1602)
- add support for the `--config` flag in the `GUNICORN_CMD_ARGS` environment variable (issue 1576, pull request 1581)

- disable `SO_REUSEPORT` by default and add the `--reuse-port` setting (issue 1553, issue 1603, pull request 1669)
- fix: installing *inotify* on MacOS no longer breaks the reloader (issue 1540, pull request 1541)
- fix: do not throw `TypeError` when `SO_REUSEPORT` is not available (issue 1501, pull request 1491)
- fix: properly decode HTTP paths containing certain non-ASCII characters (issue 1577, pull request 1578)
- fix: remove whitespace when logging header values under gevent (pull request 1607)
- fix: close unlinked temporary files (issue 1327, pull request 1428)
- fix: parse `--umask=0` correctly (issue 1622, pull request 1632)
- fix: allow loading applications using relative file paths (issue 1349, pull request 1481)
- fix: force blocking mode on the gevent sockets (issue 880, pull request 1616)
- fix: preserve leading `/` in request path (issue 1512, pull request 1511)
- fix: forbid contradictory secure scheme headers
- fix: handle malformed basic authentication headers in access log (issue 1683, pull request 1684)
- fix: defer handling of `USR1` signal to a new greenlet under gevent (issue 1645, pull request 1651)
- fix: the threaded worker would sometimes close the wrong keep-alive connection under Python 2 (issue 1698, pull request 1699)
- fix: re-open log files on `USR1` signal using `handler._open` to support subclasses of `FileHandler` (issue 1739, pull request 1742)
- deprecation: the `gaiohttp` worker is deprecated, see the *worker_class* documentation for more information (issue 1338, pull request 1418, pull request 1569)

Changelog - 2017

Note: Please see *Changelog* for the latest changes

19.7.1 / 2017/03/21

- fix: continue if SO_REUSEPORT seems to be available but fails (issue 1480)
- fix: support non-decimal values for the umask command line option (issue 1325)

19.7.0 / 2017/03/01

- The previously deprecated `gunicorn_django` command has been removed. Use the *gunicorn* command-line interface instead.
- The previously deprecated `django_settings` setting has been removed. Use the *raw_env* setting instead.
- The default value of *ssl_version* has been changed from `ssl.PROTOCOL_TLSv1` to `ssl.PROTOCOL_SSLv23`.
- fix: initialize the group access list when initgroups is set (issue 1297)
- add environment variables to gunicorn access log format (issue 1291)

- add –paste-global-conf option (issue 1304)
- fix: print access logs to STDOUT (issue 1184)
- remove upper limit on max header size config (issue 1313)
- fix: print original exception on AppImportError (issue 1334)
- use SO_REUSEPORT if available (issue 1344)
- fix leak of duplicate file descriptor for bound sockets.
- add –reload-engine option, support inotify and other backends (issue 1368, issue 1459)
- fix: reject request with invalid HTTP versions
- add `child_exit` callback (issue 1394)
- add support for eventlets _AlreadyHandled object (issue 1406)
- format boot tracebacks properly with reloader (issue 1408)
- refactor socket activation and fd inheritance for better support of SystemD (issue 1310)
- fix: o fds are given by default in gunicorn (issue 1423)
- add ability to pass settings to GUNICORN_CMD_ARGS environment variable which helps in container world (issue 1385)
- fix: catch access denied to pid file (issue 1091)
- many additions and improvements to the documentation

Breaking Change

- **Python 2.6.0** is the last supported version

Changelog - 2016

Note: Please see *Changelog* for the latest changes

19.6.0 / 2016/05/21

Core & Logging

- improvement of the binary upgrade behaviour using USR2: remove file locking (issue 1270)
- add the `--capture-output` setting to capture stdout/stderr tot the log file (issue 1271)
- Allow disabling `sendfile()` via the `SENDFILE` environment variable (issue 1252)
- fix reload under pycharm (issue 1129)

Workers

- fix: make sure to remove the signal from the worker pipe (issue 1269)
- fix: **gthread** worker, handle removed socket in the select loop (issue 1258)

19.5.0 / 2016/05/10

Core

- fix: Ensure response to HEAD request won't have message body
- fix: lock domain socket and remove on last arbiter exit (issue 1220)
- improvement: use EnvironmentError instead of socket.error (issue 939)
- add: new `FORWARDED_ALLOW_IPS` environment variable (issue 1205)
- fix: infinite recursion when destroying sockets (issue 1219)
- fix: close sockets on shutdown (issue 922)
- fix: clean up sys.exc_info calls to drop circular refs (issue 1228)
- fix: do post_worker_init after load_wsgi (issue 1248)

Workers

- fix access logging in gaiohttp worker (issue 1193)
- eventlet: handle QUIT in a new coroutine (issue 1217)
- gevent: remove obsolete exception clauses in run (issue 1218)
- tornado: fix extra "Server" response header (issue 1246)
- fix: unblock the wait loop under python 3.5 in sync worker (issue 1256)

Logging

- fix: log message for listener reloading (issue 1181)
- Let logging module handle traceback printing (issue 1201)
- improvement: Allow configuring logger_class with statsd_host (issue 1188)
- fix: traceback formatting (issue 1235)
- fix: print error logs on stderr and access logs on stdout (issue 1184)

Documentation

- Simplify installation instructions in gunicorn.org (issue 1072)
- Fix URL and default worker type in example_config (issue 1209)
- update django doc url to 1.8 lts (issue 1213)
- fix: miscellaneous wording corrections (issue 1216)

- Add PSF License Agreement of selectors.py to NOTICE (:issue: *1226*)
- document LOGGING overriding (issue 1051)
- put a note that error logs are only errors from Gunicorn (issue 1124)
- add a note about the requirements of the threads workers under python 2.x (issue 1200)
- add access_log_format to config example (issue 1251)

Tests

- Use more pytest.raises() in test_http.py

19.4.5 / 2016/01/05

- fix: NameError fileno in gunicorn.http.wsgi (issue 1178)

19.4.4 / 2016/01/04

- fix: check if a fileobject can be used with sendfile(2) (issue 1174)
- doc: be more descriptive in errorlog option (issue 1173)

Changelog - 2015

Note: Please see *Changelog* for the latest changes.

19.4.3 / 2015/12/30

- fix: don't check if a file is writable using os.stat with SELINUX (issue 1171)

19.4.2 / 2015/12/29

Core

- improvement: handle HaltServer in manage_workers (issue 1095)
- fix: Do not rely on sendfile sending requested count (issue 1155)
- fix: claridy –no-sendfile default (issue 1156)
- fix: LoggingCatch sendfile failure from no file descriptor (issue 1160)

Logging

- fix: Always send access log to syslog if syslog is on
- fix: check auth before trying to own a file (issue 1157)

Documentation

- fix: Fix Slowloris broken link. (issue 1142)
- Tweak markup in faq.rst

Testing

- fix: gaiohttp test (issue 1164)

19.4.1 / 2015/11/25

- fix tornado worker (issue 1154)

19.4.0 / 2015/11/20

Core

- fix: make sure that a user is able to access to the logs after dropping a privilege (issue 1116)
- improvement: inherit the *Exception* class where it needs to be (issue 997)
- fix: make sure headers are always encoded as latin1 RFC 2616 (issue 1102)
- improvement: reduce arbiter noise (issue 1078)
- fix: don't close the unix socket when the worker exit (issue 1088)
- improvement: Make last logged worker count an explicit instance var (issue 1078)
- improvement: prefix config file with its type (issue 836)
- improvement: pidfile handing (issue 1042)
- fix: catch OSError as well as ValueError on race condition (issue 1052)
- improve support of ipv6 by backporting urlparse.urlsplit from Python 2.7 to Python 2.6.
- fix: raise InvalidRequestLine when the line contains malicious data (issue 1023)
- fix: fix argument to disable sendfile
- fix: add gthread to the list of supported workers (issue 1011)
- improvement: retry socket binding up to five times upon EADDRNOTAVAIL (issue 1004)
- **breaking change**: only honor headers that can be encoded in ascii to comply to the RFC 7230 (See issue 1151).

Logging

- add new parameters to access log (issue 1132)
- fix: make sure that files handles are correctly reopened on HUP (issue 627)
- include request URL in error message (issue 1071)
- get username in access logs (issue 1069)
- fix statsd logging support on Python 3 (issue 1010)

Testing

- use last version of mock.
- many fixes in Travis CI support
- miscellaneous improvements in tests

Thread worker

- fix: Fix self.nr usage in ThreadedWorker so that auto restart works as expected (issue 1031)

Gevent worker

- fix quit signal handling (issue 1128)
- add support for Python 3 (issue 1066)
- fix: make graceful shutdown thread-safe (issue 1032)

Tornado worker

- fix ssl options (issue 1146, issue 1135)
- don't check timeout when stopping gracefully (issue 1106)

AIOHttp worker

- add SSL support (issue 1105)

Documentation

- fix link to proc name setting (issue 1144)
- fix worker class documentation (issue 1141, issue 1104)
- clarify graceful timeout documentation (issue 1137)
- don't duplicate NGINX config files examples (issue 1050, issue 1048)
- add *web.py* framework example (issue 1117)
- update Debian/Ubuntu installations instructions (issue 1112)
- clarify *pythonpath* setting description (issue 1080)
- tweak some example for python3
- clarify *sendfile* documentation
- miscellaneous typos in source code comments (thanks!)
- clarify why REMOTE_ADD may not be the user's IP address (issue 1037)

Misc

- fix: reloader should survive SyntaxError (issue 994)
- fix: expose the reloader class to the worker.

19.3.0 / 2015/03/06

Core

- fix: issue 978 make sure a listener is inheritable
- add *check_config* class method to workers
- fix: issue 983 fix select timeout in sync worker with multiple connections
- allows workers to access to the reloader. close issue 984
- raise TypeError instead of AssertionError

Logging

- make Logger.loglevel a class attribute

Documentation

- fix: issue 988 fix syntax errors in examples/gunicorn_rc

19.2.1 / 2015/02/4

Logging

- expose loglevel in the Logger class

AsyncIO worker (gaiohttp)

- fix issue 977 fix initial crash

Documentation

- document security mailing-list in the contributing page.

19.2 / 2015/01/30

Core

- optimize the sync workers when listening on a single interface
- add –*sendfile* settings to enable/disable sendfile. fix issue 856 .

- add the selectors module to the code base. issue 886
- add *–max-requests-jitter* setting to set the maximum jitter to add to the max-requests setting.
- fix issue 899 propagate proxy_protocol_info to keep-alive requests
- fix issue 863 worker timeout: dynamic timeout has been removed
- fix: Avoid world writable file

Logging

- fix issue 941 set logconfig default to paster more trivially
- add statsd-prefix config setting: set the prefix to use when emitting statsd metrics
- issue 832 log to console by default

Thread Worker

- fix issue 908 make sure the worker can continue to accept requests

Eventlet Worker

- fix issue 867 Fix eventlet shutdown to actively shut down the workers.

Documentation

Many improvements and fixes have been done, see the detailed changelog for more information.

Changelog - 2014

Note: Please see *Changelog* for the latest changes.

19.1.1 / 2014-08-16

Changes

Core

- fix issue 835: display correct pid of already running instance
- fix pull request 833: fix *PyTest* class in setup.py.

Logging

- fix issue 838: statsd logger, send statsd timing metrics in milliseconds
- fix issue 839: statsd logger, allows for empty log message while pushing metrics and restore worker number in DEBUG logs
- fix issue 850: add timezone to logging
- fix issue 853: Respect logger_class setting unless statsd is on

AioHttp worker

- fix issue 830 make sure gaiohttp worker is shipped with gunicorn.

19.1 / 2014-07-26

Changes

Core

- fix issue 785: handle binary type address given to a client socket address
- fix graceful shutdown. make sure QUIT and TERMS signals are switched everywhere.
- issue 799: fix support loading config from module
- issue 805: fix check for file-like objects
- fix issue 815: args validation in WSGIApplication.init
- fix issue 787: check if we load a pyc file or not.

Tornado worker

- fix issue 771: support tornado 4.0
- fix issue 783: x_headers error. The x-forwarded-headers option has been removed in c4873681299212d6082cd9902740eef18c2f14f1. The discussion is available on pull request 633.

AioHttp worker

- fix: fetch all body in input. fix issue 803
- fix: don't install the worker if python < 3.3
- fix issue 822: Support UNIX sockets in gaiohttp worker

Async worker

- fix issue 790: StopIteration shouldn't be catched at this level.

Logging

- add statsd logging handler fix issue 748

Paster

- fix issue 809: Set global logging configuration from a Paste config.

Extra

- fix RuntimeError in gunicorn.reloader (issue 807)

Documentation

- update faq: put a note on how watch logs in the console since many people asked for it.

19.0 / 2014-06-12

Gunicorn 19.0 is a major release with new features and fixes. This version improve a lot the usage of Gunicorn with python 3 by adding two new workers to it: *gthread* a fully threaded async worker using futures and *gaiohttp* a worker using asyncio.

Breaking Changes

Switch QUIT and TERM signals

With this change, when gunicorn receives a QUIT all the workers are killed immediately and exit and TERM is used for the graceful shutdown.

Note: the old behaviour was based on the NGINX but the new one is more correct according the following doc:

https://www.gnu.org/software/libc/manual/html_node/Termination-Signals.html

also it is complying with the way the signals are sent by heroku:

https://devcenter.heroku.com/articles/python-faq#what-constraints-exist-when-developing-applications-on-heroku

Deprecations

run_gunicorn, *gunicorn_django* and *gunicorn_paster* are now completely deprecated and will be removed in the next release. Use the *gunicorn* command instead.

Changes

core

- add aiohttp worker named *gaiohttp* using asyncio. Full async worker on python 3.
- fix HTTP-violating excess whitespace in write_error output

- fix: try to log what happened in the worker after a timeout, add a *worker_abort* hook on SIGABRT signal.
- fix: save listener socket name in workers so we can handle buffered keep-alive requests after the listener has closed.
- add on_exit hook called just before exiting gunicorn.
- add support for python 3.4
- fix: do not swallow unexpected errors when reaping
- fix: remove incompatible SSL option with python 2.6
- add new async gthread worker and *–threads* options allows to set multiple threads to listen on connection
- deprecate *gunicorn_django* and *gunicorn_paster*
- switch QUIT and TERM signal
- reap workers in SIGCHLD handler
- add universal wheel support
- use *email.utils.formatdate* in gunicorn.util.http_date
- deprecate the *–debug* option
- fix: log exceptions that occur after response start . . .
- allows loading of applications from *.pyc* files (#693)
- fix: issue #691, raw_env config file parsing
- use a dynamic timeout to wait for the optimal time. (Reduce power usage)
- fix python3 support when notifying the arbiter
- add: honor $WEB_CONCURRENCY environment variable. Useful for heroku setups.
- add: include tz offset in access log
- add: include access logs in the syslog handler.
- add –reload option for code reloading
- add the capability to load *gunicorn.base.Application* without the loading of the arguments of the command line. It allows you to *embed gunicorn in your own application*.
- improve: set wsgi.multithread to True for async workers
- fix logging: make sure to redirect wsgi.errors when needed
- add: syslog logging can now be done to a unix socket
- fix logging: don't try to redirect stdout/stderr to the logfile.
- fix logging: don't propagate log
- improve logging: file option can be overriden by the gunicorn options *–error-logfile* and *–access-logfile* if they are given.
- fix: don't override SERVER_* by the Host header
- fix: handle_error
- add more option to configure SSL
- fix: sendfile with SSL
- add: worker_int callback (to react on SIGTERM)

- fix: don't depend on entry point for internal classes, now absolute modules path can be given.
- fix: Error messages are now encoded in latin1
- fix: request line length check
- improvement: proxy_allow_ips: Allow proxy protocol if "*" specified
- fix: run worker's *setup* method before setting num_workers
- fix: FileWrapper inherit from *object* now
- fix: Error messages are now encoded in latin1
- fix: don't spam the console on SIGWINCH.
- fix: logging -don't stringify T and D logging atoms (#621)
- add support for the latest django version
- deprecate *run_gunicorn* django option
- fix: sys imported twice

gevent worker

- fix: make sure to stop all listeners
- fix: monkey patching is now done in the worker
- fix: "global name 'hub' is not defined"
- fix: reinit *hub* on old versions of gevent
- support gevent 1.0
- fix: add subprocess in monkey patching
- fix: add support for multiple listener

eventlet worker

- fix: merge duplicate EventletWorker.init_process method (fixes #657)
- fix: missing errno import for eventlet sendfile patch
- fix: add support for multiple listener

tornado worker

- add graceful stop support

Changelog - 2013

18.0 / 2013-08-26

- new: add `-e/--env` command line argument to pass an environment variables to gunicorn
- new: add `--chdir` command line argument to specified directory before apps loading. - new: add wsgi.file_wrapper support in async workers

- new: add `--paste` command line argument to set the paster config file
- deprecated: the command `gunicorn_django` is now deprecated. You should now run your application with the WSGI interface installed with your project (see https://docs.djangoproject.com/en/1.4/howto/deployment/wsgi/gunicorn/) for more infos.
- deprecated: the command `gunicorn_paste` is deprecated. You now should use the new `--paste` argument to set the configuration file of your paster application.
- fix: Removes unmatched leading quote from the beginning of the default access log format string
- fix: null timeout
- fix: gevent worker
- fix: don't reload the paster app when using pserve
- fix: after closing for error do not keep alive the connection
- fix: responses 1xx, 204 and 304 should not force the connection to be closed

17.5 / 2013-07-03

- new: add signals documentation
- new: add post_worker_init hook for workers
- new: try to use gunicorn.conf.py in current folder as the default config file.
- fix graceful timeout with the Eventlet worker
- fix: don't raise an error when closing the socket if already closed
- fix: fix –settings parameter for django application and try to find the django settings when using the `gunicorn` command.
- fix: give the initial global_conf to paster application
- fix: fix 'Expect: 100-continue' support on Python 3

New versionning:

With this release, the versionning of Gunicorn is changing. Gunicorn is stable since a long time and there is no point to release a "1.0" now. It should have been done since a long time. 0.17 really meant it was the 17th stable version. From the beginning we have only 2 kind of releases:

major release: releases with major changes or huge features added services releases: fixes and minor features added So from now we will apply the following versionning `<major>.<service>`. For example `17.5` is a service release.

0.17.4 / 2013-04-24

- fix unix socket address parsing

0.17.3 / 2013-04-23

- add systemd sockets support
- add `python -m gunicorn.app.wsgiapp` support

- improve logger class inheritance
- exit when the config file isn't found
- add the -R option to enable stdio inheritance in daemon mode
- don't close file descriptors > 3 in daemon mode
- improve STDOUT/STDERR logging
- fix pythonpath option
- fix pidfile creation on Python 3
- fix gevent worker exit
- fix ipv6 detection when the platform isn't supporting it

0.17.2 / 2013-01-07

- optimize readline
- make imports errors more visible when loading an app or a logging class
- fix tornado worker: don't pass ssl options if there are none
- fix PEP3333: accept only bytetrings in the response body
- fix support on CYGWIN platforms

0.17.1 / 2013-01-05

- add syslog facility name setting
- fix `--version` command line argument
- fix wsgi url_scheme for https

Changelog - 2012

0.17.0 / 2012-12-25

- allows gunicorn to bind to multiple address
- add SSL support
- add syslog support
- add nworkers_changed hook
- add response arg for post_request hook
- parse command line with argparse (replace deprecated optparse)
- fix PWD detection in arbiter
- miscellaneous PEP8 fixes

0.16.1 / 2012-11-19

- Fix packaging

0.16.0 / 2012-11-19

- **Added support for Python 3.2 & 3.3**
- Expose –pythonpath command to all gunicorn commands
- Honor $PORT environment variable, useful for deployment on heroku
- Removed support for Python 2.5
- Make sure we reopen the logs on the console
- Fix django settings module detection from path
- Reverted timeout for client socket. Fix issue on blocking issues.
- Fixed gevent worker

0.15.0 / 2012-10-18

- new documentation site on http://docs.gunicorn.org
- new website on http://gunicorn.org
- add haproxy PROXY protocol support
- add ForwardedAllowIPS option: allows to filter Front-end's IPs allowed to handle X-Forwarded-* headers.
- add callable hooks for paster config
- add x-forwarded-proto as secure scheme default (Heroku is using this)
- allows gunicorn to load a pre-compiled application
- support file reopening & reexec for all loggers
- initialize the logging config file with defaults.
- set timeout for client socket (slow client DoS).
- NoMoreData, ChunkMissingTerminator, InvalidChunkSize are now IOError exceptions
- fix graceful shutdown in gevent
- fix limit request line check

0.14.6 / 2012-07-26

- fix gevent & subproces
- fix request line length check
- fix keepalive = 0
- fix tornado worker

0.14.5 / 2012-06-24

- fix logging during daemonisation

0.14.4 / 2012-06-24

- new –graceful-timeout option
- fix multiple issues with request limit
- more fixes in django settings resolutions
- fix gevent.core import
- fix keepalive=0 in eventlet worker
- fix handle_error display with the unix worker
- fix tornado.wsgi.WSGIApplication calling error
- **breaking change**: take the control on graceful reload back. graceful can't be overrided anymore using the on_reload function.

0.14.3 / 2012-05-15

- improvement: performance of http.body.Body.readline()
- improvement: log HTTP errors in access log like Apache
- improvement: display traceback when the worker fails to boot
- improvement: makes gunicorn work with gevent 1.0
- examples: websocket example now supports hybi13
- fix: reopen log files after initialization
- fix: websockets support
- fix: django1.4 support
- fix: only load the paster application 1 time

0.14.2 / 2012-03-16

- add validate_class validator: allows to use a class or a method to initialize the app during in-code configuration
- add support for max_requests in tornado worker
- add support for disabling x_forwarded_for_header in tornado worker
- gevent_wsgi is now an alias of gevent_pywsgi
- Fix gevent_pywsgi worker

0.14.1 / 2012-03-02

- fixing source archive, reducing its size

0.14.0 / 2012-02-27

- check if Request line is too large: You can now pass the parameter `--limit-request-line` or set the `limit_request_line` in your configuration file to set the max size of the request line in bytes.
- limit the number of headers fields and their size. Add `--limit-request-field` and `limit-request-field-size` settings
- add `p` variable to the log access format to log pidfile
- add `{HeaderName}o` variable to the logo access format to log the response header HeaderName
- request header is now logged with the variable `{HeaderName}i` in the access log file
- improve error logging
- support logging.configFile
- support django 1.4 in both gunicorn_django & run_gunicorn command
- improve reload in django run_gunicorn command (should just work now)
- allows people to set the `X-Forwarded-For` header key and disable it by setting an empty string.
- fix support of Tornado
- many other fixes.

Changelog - 2011

0.13.4 / 2011-09-23

- fix util.closerange function used to prevent leaking fds on python 2.5 (typo)

0.13.3 / 2011-09-19

- refactor gevent worker
- prevent leaking fds on reexec
- fix inverted request_time computation

0.13.2 / 2011-09-17

- Add support for Tornado 2.0 in tornado worker
- Improve access logs: allows customisation of the log format & add request time
- Logger module is now pluggable
- Improve graceful shutdown in Python versions >= 2.6
- Fix post_request root arity for compatibility
- Fix sendfile support
- Fix Django reloading

0.13.1 / 2011-08-22

- Fix unix socket. log argument was missing.

0.13.0 / 2011-08-22

- Improve logging: allows file-reopening and add access log file compatible with the apache combined log format
- Add the possibility to set custom SSL headers. X-Forwarded-Protocol and X-Forwarded-SSL are still the default
- New *on_reload* hook to customize how gunicorn spawn new workers on SIGHUP
- Handle projects with relative path in django_gunicorn command
- Preserve path parameters in PATH_INFO
- post_request hook now accepts the environ as argument.
- When stopping the arbiter, close the listener asap.
- Fix Django command *run_gunicorn* in settings reloading
- Fix Tornado worker exiting
- Fix the use of sendfile in wsgi.file_wrapper

0.12.2 / 2011-05-18

- Add wsgi.file_wrapper optimised for FreeBSD, Linux & MacOSX (use sendfile if available)
- Fix django run_gunicorn command. Make sure we reload the application code.
- Fix django localisation
- Compatible with gevent 0.14dev

0.12.1 / 2011-03-23

- Add “on_starting” hook. This hook can be used to set anything before the arbiter really start
- Support bdist_rpm in setup
- Improve content-length handling (pep 3333)
- Improve Django support
- Fix daemonizing (#142)
- Fix ipv6 handling

Changelog - 2010

0.12.0 / 2010-12-22

- Add support for logging configuration using a ini file. It uses the standard Python logging’s module Configuration file format and allows anyone to use his custom file handler
- Add IPV6 support

- Add multidomain application example
- Improve gunicorn_django command when importing settings module using DJANGO_SETTINGS_MODULE environment variable
- Send appropriate error status on http parsing
- Fix pidfile, set permissions so other user can read it and use it.
- Fix temporary file leaking
- Fix setpgrp issue, can now be launched via ubuntu upstart
- Set the number of workers to zero on WINCH

0.11.2 / 2010-10-30

- Add SERVER_SOFTWARE to the os.environ
- Add support for django settings environment variable
- Add support for logging configuration in Paster ini-files
- Improve arbiter notification in asynchronous workers
- Display the right error when a worker can't be used
- Fix Django support
- Fix HUP with Paster applications
- Fix readline in wsgi.input

0.11.1 / 2010-09-02

- Implement max-requests feature to prevent memory leaks.
- Added 'worker_exit' server hook.
- Reseed the random number generator after fork().
- Improve Eventlet worker.
- Fix Django command *run_gunicorn*.
- Fix the default proc name internal setting.
- Workaround to prevent Gevent worker to segfault on MacOSX.

0.11.0 / 2010-08-12

- Improve dramatically performances of Gevent and Eventlet workers
- Optimize HTTP parsing
- Drop Server and Date headers in start_response when provided.
- Fix latency issue in async workers

0.10.1 / 2010-08-06

- Improve gevent's workers. Add "egg:gunicorn#gevent_wsgi" worker using gevent.wsgi and "egg:gunicorn#gevent_pywsgi" worker using gevent.pywsgi . **"egg:gunicorn#gevent"** using our own HTTP parser is still here and is **recommended** for normal uses. Use the "gevent.wsgi" parser if you need really fast connections and don't need streaming, keepalive or ssl.
- Add pre/post request hooks
- Exit more quietly
- Fix gevent dns issue

0.10.0 / 2010-07-08

- New HTTP parser.
- New HUP behaviour. Re-reads the configuration and then reloads all worker processes without changing the master process id. Helpful for code reloading and monitoring applications like supervisord and runit.
- Added a preload configuration parameter. By default, application code is now loaded after a worker forks. This couple with the new HUP handling can be used for dev servers to do hot code reloading. Using the preload flag can help a bit in small memory VM's.
- Allow people to pass command line arguments to WSGI applications. See: examples/alt_spec.py
- Added an example gevent reloader configuration: examples/example_gevent_reloader.py.
- New gevent worker "egg:gunicorn#gevent2", working with gevent.wsgi.
- Internal refactoring and various bug fixes.
- New documentation website.

0.9.1 / 2010-05-26

- Support https via X-Forwarded-Protocol or X-Forwarded-Ssl headers
- Fix configuration
- Remove -d options which was used instead of -D for daemon.
- Fix umask in unix socket

0.9.0 / 2010-05-24

- Added *when_ready* hook. Called just after the server is started
- Added *preload* setting. Load application code before the worker processes are forked.
- Refactored Config
- Fix pidfile
- Fix QUIT/HUP in async workers
- Fix reexec
- Documentation improvements

0.8.1 / 2010-04-29

- Fix builtins import in config
- Fix installation with pip
- Fix Tornado WSGI support
- Delay application loading until after processing all configuration

0.8.0 / 2010-04-22

- Refactored Worker management for better async support. Now use the -k option to set the type of request processing to use
- Added support for Tornado

0.7.2 / 2010-04-15

- Added –spew option to help debugging (installs a system trace hook)
- Some fixes in async arbiters
- Fix a bug in start_response on error

0.7.1 / 2010-04-01

- Fix bug when responses have no body.

0.7.0 / 2010-03-26

- Added support for Eventlet and Gevent based workers.
- Added Websockets support
- Fix Chunked Encoding
- Fix SIGWINCH on OpenBSD
- Fix PEP 333 compliance for the write callable.

0.6.5 / 2010-03-11

- Fix pidfile handling
- Fix Exception Error

0.6.4 / 2010-03-08

- Use cStringIO for performance when possible.
- Fix worker freeze when a remote connection closes unexpectedly.

0.6.3 / 2010-03-07

- Make HTTP parsing faster.
- Various bug fixes

0.6.2 / 2010-03-01

- Added support for chunked response.
- Added proc_name option to the config file.
- Improved the HTTP parser. It now uses buffers instead of strings to store temporary data.
- Improved performance when sending responses.
- Workers are now murdered by age (the oldest is killed first).

0.6.1 / 2010-02-24

- Added gunicorn config file support for Django admin command
- Fix gunicorn config file. -c was broken.
- Removed TTIN/TTOU from workers which blocked other signals.

0.6.0 / 2010-02-22

- Added setproctitle support
- Change privilege switch behavior. We now work like NGINX, master keeps the permissions, new uid/gid permissions are only set for workers.

0.5.1 / 2010-02-22

- Fix umask
- Added Debian packaging

0.5.0 / 2010-02-20

- Added configuration file handler.
- Added support for pre/post fork hooks
- Added support for before_exec hook
- Added support for unix sockets
- Added launch of workers processes under different user/group
- Added umask option
- Added SCRIPT_NAME support
- Better support of some exotic settings for Django projects

- Better support of Paste-compatible applications
- Some refactoring to make the code easier to hack
- Allow multiple keys in request and response headers

Index

R