Stopping and Restarting

This document covers stopping and restarting Apache on Unix-like systems. Windows NT, 2000 and XP users should see Running Apache as a Service [1] and Windows 9x and ME users should see Running Apache as a Console Application [2] for information on how to control Apache on those platforms.

Topics

See also

- httpd[3]
- apachectl[4]

Introduction

In order to stop or restart Apache, you must send a signal to the running `httpd` processes. There are two ways to send the signals. First, you can use the unix `kill` command to directly send signals to the processes. You will notice many `httpd` executables running on your system, but you should not send signals to any of them except the parent, whose pid is in the `PidFile`. That is to say you shouldn't ever need to send signals to any process except the parent. There are three signals that you can send the parent: `TERM`, `HUP`, and `USR1`, which will be described in a moment.

To send a signal to the parent you should issue a command such as:

```
kill -TERM `cat /usr/local/apache2/logs/httpd.pid`
```

The second method of signaling the `httpd` processes is to use the `-k` command line options: `stop`, `restart`, and `graceful`, as described below. These are arguments to the httpd [3] binary, but we recommend that you send them using the apachectl [4] control script, which will pass them through to `httpd`.

After you have signaled `httpd`, you can read about its progress by issuing:

```
tail -f /usr/local/apache2/logs/error_log
```

Modify those examples to match your `ServerRoot` and `PidFile` settings.

Stop Now

Signal: TERM

```
apachectl -k stop
```

Sending the `TERM` or `stop` signal to the parent causes it to immediately attempt to kill off all of its children. It may take it several seconds to complete killing off its children. Then the parent itself exits. Any requests in progress are terminated, and no further requests are served.

Graceful Restart

Signal: USR1

```
apachectl -k graceful
```

The `USR1` or `graceful` signal causes the parent process to *advise* the children to exit after their current request (or to exit immediately if they're not serving anything). The parent re-reads its configuration files and re-opens its log files. As each child dies off the parent replaces it with a child from the new *generation* of the configuration, which begins serving new requests immediately.

> On certain platforms that do not allow `USR1` to be used for a graceful restart, an alternative signal may be used (such as `WINCH`). The command `apachectl graceful` will send the right signal for your platform.

This code is designed to always respect the process control directive of the MPMs, so the number of processes and threads available to serve clients will be maintained at the appropriate values throughout the restart process. Furthermore, it respects `StartServers` in the following manner: if after one second at least `StartServers` new children have not been created, then create enough to pick up the slack. Hence the code tries to maintain both the number of children appropriate for the current load on the server, and respect your wishes with the `StartServers` parameter.

Users of the `mod_status` will notice that the server statistics are **not** set to zero when a `USR1` is sent. The code was written to both minimize the time in which the server is unable to serve new requests (they will be queued up by the operating system, so they're not lost in any event) and to respect your tuning parameters. In order to do this it has to keep the *scoreboard* used to keep track of all children across generations.

The status module will also use a `G` to indicate those children which are still serving requests started before the graceful restart was given.

At present there is no way for a log rotation script using `USR1` to know for certain that all children writing the pre-restart log have finished. We suggest that you use a suitable delay after sending the `USR1` signal before you do anything with the old log. For example if most of your hits take less than 10 minutes to complete for users on low bandwidth links then you could wait 15 minutes before doing anything with the old log.

> If your configuration file has errors in it when you issue a restart then your parent will not restart, it will exit with an error. In the case of graceful restarts it will also leave children running when it exits. (These are the children which are "gracefully exiting" by handling their last request.) This will cause problems if you attempt to restart the server -- it will not be able to bind to its listening ports. Before doing a restart, you can check the syntax of the configuration files with the `-t` command line argument (see httpd[3]). This still will not guarantee that the server will restart correctly. To check the semantics of the configuration files as well as the syntax, you can try starting `httpd` as a non-root user. If there are no errors it will attempt to open its sockets and logs and fail because it's not root (or because the currently running `httpd` already has those ports bound). If it fails for any other reason then it's probably a config file error and the error should be fixed before issuing the graceful restart.

Restart Now

Signal: HUP

```
apachectl -k restart
```

Sending the `HUP` or `restart` signal to the parent causes it to kill off its children like in `TERM`, but the parent doesn't exit. It re-reads its configuration files, and re-opens any log files. Then it spawns a new set of children and continues serving hits.

Users of `mod_status` will notice that the server statistics are set to zero when a `HUP` is sent.

> If your configuration file has errors in it when you issue a restart then your parent will not restart, it will exit with an error. See above for a method of avoiding this.

Appendix: signals and race conditions

Prior to Apache 1.2b9 there were several *race conditions* involving the restart and die signals (a simple description of race condition is: a time-sensitive problem, as in if something happens at just the wrong time it won't behave as expected). For those architectures that have the "right" feature set we have eliminated as many as we can. But it should be noted that there still do exist race conditions on certain architectures.

Architectures that use an on disk `ScoreBoardFile` have the potential to corrupt their scoreboards. This can result in the "bind: Address already in use" (after `HUP`) or "long lost child came home!" (after `USR1`). The former is a fatal error, while the latter just causes the server to lose a scoreboard slot. So it might be advisable to use graceful restarts, with an occasional hard restart. These problems are very difficult to work around, but fortunately most architectures do not require a scoreboard file. See the `ScoreBoardFile` documentation for a architecture uses it.

All architectures have a small race condition in each child involving the second and subsequent requests on a persistent HTTP connection (KeepAlive). It may exit after reading the request line but before reading any of the request headers. There is a fix that was discovered too late to make 1.2. In theory this isn't an issue because the KeepAlive client has to expect these events because of network latencies and server timeouts. In practice it doesn't seem to affect anything either -- in a test case the server was restarted twenty times per second and clients successfully browsed the site without getting broken images or empty documents.

URI References

[1] http://httpd.apache.org/docs-2.1/platform/windows.html#winsvc
[2] http://httpd.apache.org/docs-2.1/platform/windows.html#wincons
[3] http://httpd.apache.org/docs-2.1/programs/httpd.html
[4] http://httpd.apache.org/docs-2.1/programs/apachectl.html