

# Apache and Open-Source Development

Prepared and presented by: Ken Coar  
VP of Conference Planning,  
Apache Software Foundation and  
Senior Software Engineer,  
IBM Corporation

*Presented at the November 1999 Alternative: Linux Conference in Montréal, Québec, Canada,  
and the December 1999 Bazaar conference in New York City, New York, USA*

## **Topics:**

- Definitions
- The Apache Group versus the Apache Software Foundation
- Who Develops?
- The 'Core' Team
- The Mailing Lists
- Source Control
- Decisions
- The Release Process
- Advantages and Disadvantages
- Conclusions

You can find the latest version of this handout online at  
<<http://Golux.Com/coar/slides/>>.

## Some Definitions

### *Apache Group:*

The innermost group of developers on the Apache HTTP project; also called the *Apache core* or the *core team* of the project. Due to the fact that other projects are appearing under the aegis of the Apache Software Foundation, this name can be somewhat confusing – but it refers specifically to the HTTP server project.

### *Apache HTTP server:*

The original Apache project, a Web server originally based on the popular open-source server from NCSA.

### *Apache Software Foundation:*

A not-for-profit membership-based company incorporated to support the Apache HTTP server project and others, and to further the aims of the open-source movement by providing infrastructural, legal, and other assistance for open-source projects.

### *commit-then-review (CTR):*

A methodology of applying changes to a master set of source code, involving the assumption of goodness. Changes are deemed inherently acceptable and are applied, and tested and reviewed afterward. They remain in effect unless and until someone raises an objection, at which point they may be reverted. Compare *review-then-commit*.

### *core group or core team:*

The collection of developers most closely involved in a project; the decision-makers. For the Apache HTTP server project, the core team is also called the Apache Group.

### *CVS:*

Acronym for **C**oncurrent **V**ersioning **S**ystem, a software package that allows multiple people to work on private copies of files and check in changes in a controlled by asynchronous manner. CVS is well-known for its technique of *lazy locking*. The alternative, *strict locking*, mandates exclusive access to files under development; while one person has a file checked out, no-one else can make changes to it until that person either checks it back in or releases the lock, abandoning any changes. CVS is heavily used in distributed open-source software projects.

### *CVS access:*

In terms of Apache, having “CVS access” means that being able to commit changes directly to the master repository sources without requiring review by anyone else. Whether doing so is procedurally correct depends upon the human rules in effect; see *commit-then-review* and *review-then-commit*.

### *meritocracy:*

A system of management in which the amount of access and participation allowed are based upon the opinion of one’s peers. One of the underlying ideas of the Apache Group meritocracy is ‘the more you do, the more you’re allowed to do.’ The more you participate, the more respect and ‘merit’ you earn from your peers, and the more they permit you to do.

### *review-then-commit:*

A change application methodology that involves changes being proposed, discussed, considered, and reviewed before being applied. Prior to February 1998, most of the development done in the Apache HTTP server project followed this method, with proposed changes requiring three ‘yea’ votes before being allowed into the source. Compare *commit-then-review*.

## The Apache Group *versus* the Apache Software Foundation

No, that title doesn't reflect an adversarial relationship; it's meant to indicate that the two are not the same entity.

For the last several years, a piece of software known as 'Apache' has been getting more and more visibility. Starting as an obscure minor refit of the then-popular NCSA Web server package, Apache has since steadily grown in mindshare and marketshare. What *is* Apache, and how is it doing this?

Well, in a word (or a few words), it's a labour of love. It's a bundle of software that a number of people work on, from minor twiddling to massive hacking, mostly on their own time. Each person working on it tends to have a particular area of interest, or perhaps a few areas, and no one person is responsible for the Big Picture. Development is done by a sort of empowered committee method; people propose changes they want to make, the changes are discussed, refined, and implemented – or possibly abandoned or discarded.

For most of its existence, there have been fewer than two dozen people seriously working on the software at any one time. Despite the chaotic appearance of the development method, the Apache server has nevertheless achieved market prominence: more than half of the Web servers on the Internet use the free Apache server rather than one of the commercial alternatives.

This proved to be an embarrassment of riches; while the developers were gratified at the success of their 'midnight hack,' it attracted a lot of attention. Enough attention, in fact, that a lot of time was having to be spent dealing with questions about the software licence, the name, and other matters – time which obviously couldn't be spent on development. Also, with so many eyes focussed on the project, there was potential for legal difficulties: being developers rather than lawyers, it was quite possible that the Apache Group (as the core team is called) might inadvertently commit some sort of trademark trespass or infringe some patent claim.

So, in March of 1999, the people who were the primary developers of the software (the 'core team') incorporated a not-for-profit company called the Apache Software Foundation. This had several advantages:

- The members of the Apache Group who were interested in the administrative, business, and political side of the Apache project could be empowered to take care of that side of things, and the people who wanted nothing more than the freedom to code didn't have to be bothered with discussions and votes on things that didn't interest them.
- With the ownership and control of the software assets transferred to the Apache Software Foundation, individual developer liability was greatly lessened.
- The name 'Apache' could be developed as a brand, extending to cover other open-source projects in addition to the HTTP server, and the Foundation could provide such projects with the same or similar support as the original project had and has (software infrastructure, mailing lists, legal protection, *et cetera*).

So while this session describes the way in which the development of the Apache HTTP server software is developed, it's no longer quite accurate to call it the 'Apache development model' – because other projects under the Foundation are free to use other rules, processes, and methods. The Foundation exists to support the projects in their endeavours, not to impose particular methodologies on them. It's the openness of the project that's important, and pretty much whatever works to accomplish that is acceptable.

## Who Develops?

The primary development of the Apache HTTP server is done by a relatively small number of developer; however, the server is the results of contributions from hundreds of people all over the world. So the real answer to 'who are the developers' is 'everyone who wants to be.'

In many cases contributions are small, or a contributor submits one item and is then never heard from again, but the server has benefited from those small submissions just as surely as from the work done by the core developers who put in hundreds of hours *per* year on the project.

There are several levels of participation in the project, with rather blurry lines between them. From the most to the least, these fall into the following basic groupings:

1. Membership in the 'core' development team, which is synonymous with the Apache Group
2. People with direct access (called *CVS* or *commit access*) to all of the master source repositories
3. People with CVS access to some subset of the repositories
4. People who participate in the mailing lists and have earned enough respect for their votes to be counted
5. Everyone else

At all levels a participant has the ability to 'check out' a copy of the source repositories; it's only at the top three, though, that changes can be checked back in without having to go through someone else.

This arrangement has been called a 'meritocracy;' the more you do, the more you're allowed to do. Access to each successive level of participation is controlled by the opinions of your peers. The more you submit, and the more your submissions are considered to be of value, the more 'merit' you acquire. Accumulate enough merit and you'll be admitted to the next level of participation. Almost all such decisions are made by the core team.

## The 'Core' Team

The innermost circle of involvement in the Apache server project is comprised of the people who have shown the most dedication to it, either through actual code submitted or by advocacy, infrastructure support, or other types of contributions.

The core team originally consisted of the project's founders, but some of those have moved on and new people have discovered Apache and become involved, so the membership has changed over time.

## The Mailing Lists

Since Apache development is so highly distributed, with lots of work being done in Europe and Oceania in addition to the various time-zones in the United States, real-time communications just don't work very well. *Someone* is going to be asleep or close to it at almost any possible time.

The solution currently in use by the Apache project is to have most everything happen in electronic mail. There are a few mailing lists dedicated to the project (described below), and even the most impatient participants learn quickly to allow at least 24 hours to pass before assuming any sort of consensus or conclusion. Discussions can often span several days or even weeks as people who were offline come back and join in, possibly re-opening a discussion that others thought was concluded.

In addition to the main discussion list, there are a couple of others which exist solely to keep people apprised of updates as they happen, such as the submission of problem reports or notifications of changes made to the master source repositories.

Subscription information for all of the public mailing lists described here can be found by following links from the main development Web site, <http://dev.apache.org/>.

## ***Development and discussion***

There are two discussion lists associated with the Apache HTTP server project: a public technical list called `new-httpd`, and the private `apache-core` list which is intended for discussions by the core team about the project itself rather than about technical issues.

The `new-httpd` list can achieve amazingly high traffic levels; at peaks of development activity (or controversy) it may involve hundreds of messages *per* day. Even during the quietest of times there are usually at least half a dozen messages in any 24-hour period.

The list is definitely for serious technical discussion; there is not a lot of tolerance exhibited toward newcomers seeking consulting help with server operation. As stated on the Web site, <http://www.apache.org/>, other avenues exist for such questions – primarily the Usenet newsgroups.

Even messages containing legitimate technical development-related content aren't guaranteed a welcome; the list could easily be described as 'high-octane.' Someone new who posts on a subject that has been discussed (and concluded) several times before, or who weighs into the middle of a vigorous technical discussion with something ill-considered or poorly presented, is likely to get 'flamed.' Even messages that pass the flame-bait test might be given little credence at first.

This is because the community of participants on the `new-httpd` list is just that – a community. The people who have been there for a while are familiar with each other, and have developed respect for each other's opinions; newcomers are generally greeted with reticence until they have established a bit of a presence and the long-term residents feel comfortable with them.

This makes it sound as though new contributors aren't welcome, but that's actually very far from the truth. The project is *always* glad to get more help – but would-be contributors need to show that they're there for the course before they'll be taken seriously. That means proving themselves by showing awareness of current and past discussions (through lurking for a while and reviewing the mail archives), technical ability, perseverance, and the courage to stand up for their opinions. A would-be contributor who can do this steadily will almost certainly gain acceptance within a couple of months.

So: If you would be an Apache developer, two of the things you should do are to become at least marginally familiar with the mail archives (see <http://dev.apache.org/>), sub subscribe to the `new-httpd` list and lurk on it for two or three weeks before posting anything yourself.

There are a couple of other lists maintained by Covalent Technologies, Inc., that are intended for Apache users and module writers respectively, but they're quite low-volume, frequently with several days passing between messages.

## ***Bug reports***

Another of the Apache-related mailing lists is quite simply a means of keeping people informed about bug reports submitted against the software (see <http://bugs.apache.org/>). This is essentially a read-only list;

no discussion takes place on it. As new reports come in, or existing ones are updated, the information is posted to the subscribers.

Replies to the messages from the bugdb, as it's called, will go only to the mailing list, which typically doesn't benefit anyone. If a special address is included on the reply, however, the text will be attached to the appropriate report – which will cause a new update to be sent to the list, and to the bug submitter as well. This is one of the two ways of communicating (such as asking for more information) with bug submitters; the other involves updating the bug report (called a *PR*) directly through a special Web page.

Most if not all of the bugdb list subscribers are also on the `new-httpd` list, which explains why discussions about particular PRs might seem to suddenly appear there. To readers who are both lists, the continuity is obvious; to those who are only on the `new-httpd` list, though, the origin of such threads might seem mysterious.

## Source changes

Another 'read-only' notification list is used to keep interested parties informed of changes to the master sources. Any time a modification is committed to the repository, a summary message showing what was changed is sent to this mailing list.

As with the bugdb list, most of the people on the CVS update list are on `new-httpd` – and discussions threads concerning source changes might seem to appear out of the blue for the same reason bug report-related message do. In fact, since the source tends to be changed more frequently than bugs get reported, and the changes are more likely to be controversial, this happens much more frequently than for PRs.

## 'Core' list

The `apache-core` mailing list isn't open to the public, being reserved for private discussions by the core team. Discussions on the list are *supposed* to be non-technical in nature, such as discussing whether to grant someone additional access or discuss sensitive material (such as security vulnerability reports), but occasionally this rule gets broken and a technical thread breaks out. When someone notices that this has happened, though, the discussion is usually then moved to the `new-httpd` list.

## Source Control

Since the Apache HTTP server project is all about open source software development, there really needs to be a way in which the source itself is managed. In 1996 the decision was made by the Apache Group to move the source into a repository managed by the CVS software tool, which supports asynchronous distributed development very well. CVS is based on the underlying RCS (revision control system) software, but you don't need to know RCS in order to use CVS.

Like other source control systems, CVS maintains a history, so you can get a copy of a file as it existed on a particular date or as of a specific release, as well as see what all the changes have been, when they were made, by whom, and why. For an example of these latter capabilities, visit the development Web site at

<http://dev.apache.org/> and follow the links for 'Web-based view of the CVS repository.'

This history information is available to one and all, as is access to the source code itself. The ability to actually make changes is reserved to just a few people, however. For people who want to track the development of the code in close to real time, there are `rsync` and `anoncvs` access paths available; see the <http://dev.apache.org/> Web site for details. For more information about CVS itself, see the pages at <http://www.cyclic.com/>.

## **Review-Then-Commit**

In a development project like the Apache HTTP server, in which there is no central control and changes can be made by a number of peers working independently, *some* sort of coordination is necessary. Otherwise the potential exists for people to keep making conflicting changes or otherwise interfere with each other.

One coordination method is called *review-then-commit* (RTC), and involves the changes being proposed by the would-be changer, discussed by the development team, and then getting applied to the sources when some sort of agreement is reached.

In the Apache project, this was the normal mode of operation from at least 1996 through late 1997 (quite a long period, in “Internet time”). Someone who wanted to make a change would work out the details, test it, and submit it in the form of a ‘patch’ to the mailing list. Other developers would then apply the patch to their own systems and test the result, and then either suggest alterations or approve it as supplied. Or, if they disliked the entire idea, they might veto it without bothering to test it.

At any rate, in order for a change to be made to the master sources under the RTC system it needs to first garner approval from the development group or at least a significant fraction of it. (See the section on ‘Voting’ later in this handout.)

One of the advantages of RTC is that only those changes that are really supported and approved by the development group get applied. A disadvantage is that the process is generally quite time-consuming, and doesn’t encourage innovation very well.

## **Commit-Then-Review**

In the Autumn of 1998, about halfway through the development cycle of the Apache 1.3 release, enough of the developers were fretting under the approval delay in the commit-then-review process that the Apache Group decided to try an alternative: *commit-then-review*. This operates exactly as you might expect: if one of the core developers wants to make a change to the source, he just goes ahead and does it. Of course, some niceties and etiquette are observed; if a change is likely to be controversial, it’s supposed to be discussed on the mailing list first; only documentation changes and bug fixes have complete immunity to that requirement.

This method shifts the roles and responsibilities of the development team somewhat. Under the review-then-commit system, a change that no-one bothered to examine would never get approved and would silently die without being implemented (unless its proposer nagged until a decision was made). Depending upon workloads and interests, this might happen with varying frequency. Under commit-then-review, however, there is somewhat more of an onus on the developers to actually examine the changes being made – because they’ve *been* made, and they’ll go into the next release unless someone finds a problem with or objection to them.

The main advantage to the CTR process is that it tends to speed up development – individuals don’t ‘hang fire’ waiting for changes to be approved; they commit them and then move on. A disadvantage, if you can call it that, is the need for increased vigilance on the part of the development team to ensure that the changes being made are, in fact, good and won’t lead to problems later on.

## **Decisions**

Without some sort of executive who can say what gets done and what doesn’t, how are decisions made in the Apache framework?

The answer is: mostly through consensus drawn from email discussions. Such discussions (often called ‘threads’ in the email world) may take from a couple of days to a couple of months to arrive at a generally-accepted conclusion, and some may never reach that stage.

When the various positions on a particular topic seem to have been identified, typically someone will call for a vote, and people on the mailing list will express their opinions and alignments.

This is where an odd grey area in the ‘levels of access’ list becomes apparent. Anyone subscribed to the `new-httpd` mailing list is free to express an opinion by voting, but its vote will be ignored in the tally unless the person has ‘arrived’ and is recognised as a serious contributor. If someone votes without having reached this state, it is likely to be the recipient of several private email messages apprising it of the inappropriateness of its action. There’s no clear hard-and-fast rule about when this condition will change; it has to be sensed from noticing when one’s comments are actually taken seriously on the mailing list and one’s vote is actually counted. As a rule of thumb, this typically requires at least a couple of months’ worth of discussion contribution on the list.

## Voting

When it comes to voting, the Apache environment uses a peculiarly geekish notation. If someone is in favour of an idea, it expresses that by replying with “+1” in a mail message. If it is opposed, this is indicated by a message containing “-1”. In the case of no strong position, someone might vote “0,” “+0,” and “-0,” signifying “no opinion,” “sorta like the idea,” and “sorta dislike the idea.” Depending upon the subject of the vote, the issue will be closed when a sufficient number of opinions have been tallied.

For technical decisions, an issue is considered approved as soon as three in-favour (“+1”) votes are registered. Other issues typically go by a more seat-of-the-pants method, whether a clear majority is for or against the idea.

The details of the voting guidelines are available on the Apache development Web site; go to <http://dev.apache.org/> and look for links for ‘guidelines’ or ‘voting.’

## Vetoes

For technical issues, a special rule applies that can block the acceptance or adoption of the issue under discussion. Anyone who is empowered to vote is allowed to exercise a veto, represented by a vote of “-1”. **There is no appeal of a veto; it blocks the issue until withdrawn by the person who exercised it.** A veto is only valid if it’s accompanied by a technical justification.

The only way past a veto is to convince the vetoer to withdraw it. As you might expect, vetoing a change is a very serious act and is not done lightly; far more issues die through lack of acquiring the requisite number of favourable votes than do because they’re vetoed.

The “-1” notation means “I veto this” in one context, and “I vote ‘nay’” in others. The only difference is the type of issue being voted.

## The Release Process

New versions of the Apache HTTP server software aren’t released according to any calendar schedule; instead, they are made available when there’s consensus among the developers that it’s “ready.” Every now and then, someone on the development mailing list notices that, “gee, it’s been a long time since we did a release; the `src/CHANGES` file has about fifty new entries since the last one.” After a few weeks of discussion, general agreement will probably form that yes, it’s about time for a new release, and someone will volunteer to be the release manager.

Of course, sometimes new releases come out in short order, if a major problem was discovered in the just-released code. This happens about every third or fourth release.

The release manager has the responsibility for getting agreement on the release schedule, monitoring new commits get made to make sure they're not too controversial, contacting the testers' mailing lists, and building the release. If all goes well, and the release is considered valid, someone – possibly the release manager, but usually not – will announce the availability of the new package.

And then people start hacking on the code as usual, building up toward the next release.

## **Tagging, Rolling, Testing, and Releasing**

Once the code has been 'frozen' for packaging, the release manager has to go through a series of steps to actually construct the package and make it available. The details of the process can be found on the <http://dev.apache.org/> Web site, but here are the basics:

1. Various version numbers embedded in the source code need to be updated, such as changing the version "Apache 1.3.10-dev" to "Apache 1.3.10". (The "-dev" suffix marks a snapshot of the code as being in pre-release development for the named version.)
2. The entire set of source files need to be "tagged" – that is, the latest revision of each file gets marked as being the one used to make this particular release. The version numbers in the master sources are then advanced, as "Apache 1.3.10" becoming "Apache 1.3.11-dev". In theory, normal development could resume at this point, but that's discouraged for reasons explained below.
3. The tagged version of the source needs to be 'exported' to form the directory tree used to build the release package. 'Exporting' just means that copies of the files are extracted from the CVS repository without any revision control information, so they're just plain files in a just plain directory tree.
4. Using the exported sources, the software package gets built using a standard set of options. The package is then packed into a *tarball*, or `tar` archive file. The tarballs (compressed using a couple of different methods) are signed by the release manager with PGP, and checksummed with md5.
5. The tarballs and the signatures are put into a special Web location, and the developers and testers are invited to download it and check it out.
6. If no serious problems are encountered, the tarballs and signatures are moved into a public distribution location (<http://www.apache.org/dist/>), and the Web site is updated. Twenty-four hours later, the new release is announced on various mailing lists and Usenet newsgroups. (The delay is to allow the various world-wide mirrors of the Apache site to pick up the changes and the new release files.)

Any serious "showstopper" problem occurring in steps 4 or 5 will invalidate the release, and the release manager will need to coordinate the fixing of them and then start over again. This means that release version numbers may skip from time to time, such as from 1.3.4 to 1.3.6, or from 1.3.6 to 1.3.9 (both of which actually happened).

Generally, the release manager calls for a complete hold on new development from the time the files are tagged until the release is successfully finalised and made available. This allows the only changes from the freeze until the final release to be directly related to fixing known serious problems, rather than potentially introducing new issues through unrelated modifications.

## Advantages and Disadvantages

No process is perfect or completely free from warts. The methodology the Apache HTTP server project uses is, of course, no exception. Some of the issues that have been noticed include:

- With only a small number of people able to approve or actually make changes, the software has a tendency to reflect the opinions of a few experts rather than a lot of end-users. On the other hand, this also has a tendency to improve quality.
- Changes that get made are generally approved by the development team – but in the case of any uncertainty, it may take a lot of long, tedious discussion to get to the point of approval.
- The Apache HTTP server development community consists of a lot of highly opinionated individuals, and discussions are sometimes acrimonious. Coupled with the strong bond amongst the community members, this means that it's not easy for new people to break into and join the community.
- Because of the very limited size of the development team, acceptance as part of it is a sort of credential recognised in the broader open-source community. Being known as an “Apache developer” is no bad thing.

Other open-source projects do things differently, such as being less restrictive of who has access to the source repository (PHP), or possibly having a single person who is the final arbiter of what gets accepted (Linux, `mod_dav`, `mod_ssl`). The Apache HTTP server project recently (November 1999) decided to try an experiment by moving the Web server documentation into a separate CVS module and ‘lowering the bar’ as to who has access to it (*i.e.*, letting more people be able to modify it). The last experiment of this sort involved the test of whether *commit-then-review* was a viable development alternative, and it was successful.

## Conclusion

Weird and arcane though the Apache HTTP server project's development system may seem, we must be doing *something* right, or else we wouldn't have over half the Internet Web server market (see <http://www.netcraft.com/Survey/>).

Want to know more? Check out the Apache Software Foundation Web site <http://www.apache.org/>, the Apache HTTP Server project Web site <http://www.apache.org/httpd.html>, and the development Web site <http://dev.apache.org/>. Consider joining the mailing lists, or at least reading the archives (described at the development site).

And consider joining us at the ApacheCon 2000 conference in Orlando, Florida, USA on March 8-10<sup>th</sup> 2000! Find out more at <http://ApacheCon.Com/>.