Apache Forrest project guidelines

Table of contents

1 The mission of Apache Forrest 2
2 Open development 2
3 Roles and responsibilities 2
4 Project Management Committee (PMC) 2
4.1 Quarterly reports to ASF Board 3
4.2 Electing new committers and PMC members 4
5 Decision making 4
5.1 Voting 4
5.2 Types of approval 5
5.3 Vetoes 5
5.4 Actions 6
5.5 Voting timeframes 7
5.6 Voting procedure 7
5.7 Ultimatum and breakdown 7
6 Communication channels 7
7 Code management 8
8 Contribution and acknowledgement 8
9 Development procedure 9

This document provides the guidelines under which the Apache Forrest project operates. It defines the roles and responsibilities, who may vote, how voting works, how conflicts are resolved, etc. Apache Forrest is a project of The Apache Software Foundation (ASF) which is a non-profit corporation. As with all such organisations there are some procedures to be followed. The ASF website explains the operation and background of the ASF. These project guidelines supplement that ASF documentation. Normally these guidelines are not needed - the project just gets on with its day-to-day operation - but they enable all people to understand how the project operates.

1. The mission of Apache Forrest

The creation and maintenance of open-source software for distribution at no charge to the public, with the purpose of generation of aggregated multi-channel documentation maintaining a separation of content and presentation.

2. Open development

Forrest is typical of Apache projects, in that it operates under a set of principles that encourage open development. There is no clear definition (perhaps that is part of it) and it is ever-evolving. Each ASF project is different in its own way - there is healthy diversity rather than uniformity across all projects. The main principles are to facilitate open collaborative development, with respect for others; to ensure that there is a healthy community (even to give community issues higher priority than code issues); to use a consensus-based approach; to ensure that each contributor is recognised and feels a productive part of the community; to encourage diversity; to make the project a nice place to be.

Each project, as part of the resolution that formed its Project Management Committee (PMC), creates its set of project guidelines intended to encourage open development and increased participation. These facilitate the project to conduct its main charge, which is the creation and maintenance of open-source software for distribution at no charge to the public with the purpose of its mission.

For more information about the way that Apache projects operate, please refer to the ASF foundation and ASF developer sections of the ASF website, including the ASF ByLaws and the How it works document, the FAQs about the Foundation, and the Incubator project.

3. Roles and responsibilities

The meritocracy enables various roles as defined in the How it works document.

user | developer | committer | PMC member | ASF member

The Apache Forrest committers and PMC members are listed.

4. Project Management Committee (PMC)

The Apache Forrest project was established in January 2002 and became a top-level project in May 2004. The Project Management Committee (PMC) was created by a resolution of the board of the Apache Software Foundation. See explanation of the role of the PMC in that resolution and also the ASF Bylaws and How-it-works and this mail thread.

At Forrest, the group of PMC members essentially equates to the group of committers. We encourage

all committers to be PMC members. See explanation below. See the "who we are" page for explanation of why some committers from the old project are not PMC members.

PMC members can be as active as they choose, with no pressure from the project. People can be quiet and speak up occasionally when they see a topic that motivates them enough to contribute to the discussion or to cast a vote. Individual PMC members do not need to be involved in every aspect of the project. As a group, the PMC will maintain sufficient oversight.

The responsibilities of the PMC include:

- Be familiar with these project guidelines, and the ASF Bylaws, and with the ASF documentation and procedures in general.
- Keep oversight of the commit log messages and ensure that the codebase does not have copyright and license issues, and that the project is heading in the desired direction.
- Keep oversight of the mailing lists and community to ensure that the open development ideals are upheld.
- Resolve license disputes regarding products of the project, including other supporting software that is re-distributed.
- Decide what is distributed as products of the project. In particular all releases must be approved by the PMC.
- Guide the direction of the project.
- Strive for and help to facilitate a harmonious productive community.
- Nominate new PMC members and committers.
- Maintain the project's shared resources, including the codebase repository, mailing lists, websites.
- Speak on behalf of the project.
- Maintain these and other guidelines of the project.

The PMC does have a private mailing list on which it can discuss certain issues. However this list is rarely used and every effort is made to conduct all discussion on the public mailing lists.

Membership of the PMC is by invitation only and must receive consensus approval of the PMC members.

The actual list of PMC members is in the SVN "committers" repository at /board/committee-info.txt and is reflected at the "who we are" page.

A PMC member is considered "emeritus" by their own declaration, e.g. perhaps personal reasons. Please send a note to the PMC private mail list and we will follow up to request acknowledgement of the Board. An emeritus member may request reinstatement to the PMC. Such reinstatement is subject to consensus approval of the PMC members. Membership of the PMC can be revoked by unanimous consensus of PMC members (other than the member in question).

The chair of the PMC is appointed by the Board and is an officer of the ASF (Vice President). The chair has primary responsibility to the Board, and has the power to establish rules and procedures for the day-to-day management of the communities for which the PMC is responsible, including the composition of the PMC itself. The chair reports to the board every three months about the status of the project. The PMC may consider the position of PMC chair annually and may recommend a new chair to the board. Ultimately, however, it is the board's responsibility who it chooses to appoint as the PMC chair. See further explanation of the role of the chair in the ASF Bylaws and the PMC FAQ

4.1. Quarterly reports to ASF Board

Every three months, it is the responsibility of our PMC chair to send a report to the ASF Board. This is

mainly concerned with the status of our community, but can also include the technical progress. Further details are in the "committers" SVN in the /board/ directory.

The minutes are available for each board meeting. Our reporting schedule is: Feb, May, Aug, Nov.

4.2. Electing new committers and PMC members

When we see new people who are committed and consistent, we will discuss each case to ensure that the PMC is in agreement. See the list of qualities that we look for. We conduct the vote on the private PMC mailing list to enable a frank discussion and so that we do not conduct public discussions about people.

In most cases we will be inviting people to go straight from developer to PMC member, i.e. they simultaneously become committer and PMC member. We always want new committers to also be PMC members. Otherwise they do not have a binding vote and so we would create classes of committers. Another issue is indemnification. However, when we invite a new committer we do let them choose not to be on the PMC, though we do not encourage that.

However, there may be extraordinary cases where we want limited work-related commit access and so the committer is not also a PMC member (e.g. perhaps temporary access for GSoC). This will be resolved during the discussion and vote.

PMC members can also see further notes about the process of electing new people.

5. Decision making

Different types of decisions require different forms of approval. For example, the previous section describes several decisions which require "consensus approval". This section defines how voting is performed, the types of approval, and which types of decision require which type of approval.

Most day-to-day operations do not require explicit voting - just get on and do the work. See the "Lazy approval" type described below.

5.1. Voting

Certain actions and decisions regarding the project are made by votes on the project development mailing list. Where necessary, PMC voting may take place on the private PMC mailing list.

Votes are clearly indicated by subject line starting with [VOTE]. Discussion and proposal should have happened prior to the vote. Voting is carried out by replying to the vote mail. See voting procedure below. Votes are expressed using one of the following symbols:

+1	"Yes," "Agree," or "the action should be performed." In general, this vote also indicates a willingness on the behalf of the voter to assist with "making it happen".
+0	This vote indicates a willingness for the action under consideration to go ahead. The voter, however will not be able to help.
-0	This vote indicates that the voter does not, in general, agree with the proposed action but is not concerned enough to prevent the action

	going ahead.
-1	This is a negative vote. On issues where consensus is required, this vote counts as a veto. All vetoes must contain an explanation of why the veto is appropriate. Vetoes with no explanation are void. It may also be appropriate for a -1 vote to include an alternative course of action.
abstain	People can abstain from voting. They can either remain silent or express their reason.

All participants in the project are encouraged to show their preference for a particular action by voting. When the votes are tallied, only the votes of PMC members are binding. Non-binding votes are still useful to enable everyone to understand the perception of an action by the wider community.

Voting can also be applied to changes made to the project codebase. These typically take the form of a veto (-1) in reply to the commit message sent when the commit is made.

5.2. Types of approval

Different actions require different types of approval:

Consensus approval	Consensus approval requires 3 binding +1 votes and no binding vetoes.
Lazy majority	A lazy majority vote requires 3 binding +1 votes and more binding +1 votes than -1 votes.
Lazy approval	An action with lazy approval is implicitly allowed unless a -1 vote is received, at which time, depending on the type of action, either lazy majority or consensus approval must be obtained.
2/3 majority	Some actions require a 2/3 majority of PMC members. Such actions typically affect the foundation of the project (e.g. adopting a new codebase to replace an existing product). The higher threshold is designed to ensure such changes are strongly supported. To pass this vote requires at least 2/3 of the votes that are cast to be +1.
Unanimous consensus	All of the votes that are cast are to be +1 and there can be no binding vetoes (-1).

5.3. Vetoes

A valid veto cannot be over-ruled, it can only be withdrawn by its issuer. Any veto must be accompanied by reasoning and be prepared to defend it.

The validity of a veto, if challenged, can be confirmed by anyone who has a binding vote. This does not necessarily signify agreement with the veto - merely that the veto is valid. In case of disputes about whether a veto is valid, then opinion of the PMC chair is final.

If you disagree with a valid veto, then you must engage the person casting the veto to further discuss

the issues. The vetoer is obliged to vote early and to then work with the community to resolve the matter.

If a veto is not withdrawn, the action that has been vetoed must be reversed in a timely manner.

5.4. Actions

This section describes the various actions which are undertaken within the project, the corresponding approval required for that action, and those who have binding votes over the action.

Action	Description	Approval	Binding Votes
Code change	A change made to a codebase of the project by a committer. This includes source code, documentation, website content, etc.	Lazy approval	PMC members
Release plan	Defines the timetable and actions for a release. A release plan cannot be vetoed (hence lazy majority).	Lazy majority	PMC members
Product release	When a release of one of the project's products is ready, a vote is required to accept the release as an official release of the project. A release cannot be vetoed (hence lazy majority). The Release Manager might choose to get an issue resolved or re-schedule, but that is their decision.	Lazy majority	PMC members
Adoption of new codebase	When the codebase for an existing, released product is to be replaced with an alternative codebase. If such a vote fails to gain approval, the existing code base will continue. This also covers the creation of new sub-projects within the project.	2/3 majority	PMC members
New committer	When a new committer is proposed for the project.	Consensus approval	PMC members
New PMC member	When a new member is proposed for the PMC.	Consensus approval	PMC members

Reinstate emeritus member	An emeritus PMC member can be reinstated.	Consensus approval	PMC members (excluding the member in question)
Committer removal	When removal of commit privileges is sought.	Unanimous consensus	PMC members (excluding the committer in question if a member of the PMC)
PMC member removal	When removal of a PMC member is sought. See also section 6.5 of the ASF Bylaws whereby the ASF Board may remove a PMC member.	Unanimous consensus	PMC members (excluding the member in question)

5.5. Voting timeframes

Votes are normally open for a period of one week to allow all active voters time to consider the vote. If the vote has not achieved a quorum (the chair decides if sufficient people have voted), then it can be extended for another week. If still no quorum, then the vote fails, and would need to be raised again later. Votes relating to code changes are not subject to a strict timetable, but should be made as timely as possible.

Be careful about holidays when calling a vote. This is hard when we do not know customs in every part of the world. So if someone knows that there is a problem with the vote timing, then please say so.

5.6. Voting procedure

Discussion about the topic would have already happened in a [Proposal] email thread to express the issues and opinions. The [Vote] thread is to ratify the proposal if that is felt to be necessary.

The instigator sends the Vote email to the dev mailing list. Describe the issue with no ambiguity and in a positive sense. Define the date and time for the end of the vote period.

Votes are expressed by replying email using the voting symbols defined above. Voters can change their vote during the timeframe. At the end of the vote period, the instigator tallies the number of final votes and reports the results.

5.7. Ultimatum and breakdown

For breakdown situations and those requiring unanimous consensus, if this consensus cannot be reached within the extended timeframe, then the Board expects the chair to act as the officer of the Foundation and make the ultimate decision.

6. Communication channels

The primary mechanism for communication is the mailing lists. Anyone can participate, no matter what their time zone. A reliable searchable archive of past discussion is built. Oversight is enabled. Many eyes ensures that the project evolves in a consistent direction.

All decisions are made on the "dev" mailing list.

The main channel for user support is the "user" mailing list. As is usual with mailing lists, be prepared to wait for an answer.

Occasionally we will use other communication channels such as IRC. These are used only for a specific purpose and are not permanently available. This policy ensures that solutions are available in the mailing list archives and enables people to respond at whatever time that they choose. Permanent IRC channels are poor from a community-building point-of-view, as they tend to create time-zone based cliques. So we don't.

Similarly, private discussions are discouraged. The rest of the community would not benefit from the understanding that is developed. Off-list discussions put too much load on overworked volunteers.

7. Code management

The term "patch" has two meanings: Developers provide a set of changes via our Issue Tracker marked for inclusion, which will be applied by a committer. Committers apply their own work directly, but it is still essentially a patch.

We use the Commit-then-review method for decision-making about code changes. Please refer to that glossary definition. Note that it does not preclude the committer from making changes to patches prior to their commit, nor mean that the committer can be careless. Rather it is a policy for decision-making.

There is an important issue where both developers and committers need to pay special attention: "licenses". We must not introduce licensing conditions that go beyond the terms of the Apache License. If such issues do creep in to our repository, then we must work as quickly as possible to address it and definitely before the next release.

There are some other problem areas: What should a committer do if the patch is sloppy, containing inconsistent whitespace and other code formatting, which mean that actual changes are not easily visible in the svn diff messages. What if there is poorly constructed (yet working) xml or java code? What if the new functionality is beyond the scope of the project? What if there is a better way to do the task? What if the patch will break the build, thereby preventing other developers from working and causing an unstable trunk?

The committer has various options: ask the developer to resubmit the patch; change the patch to fix the problems prior to committing; discuss the issues on the dev list; commit it and then draw attention to the issues so that the rest of the community can review and fix it. A combination of these options would appear to be the best approach. Please aim to not break the build, or introduce license problems, or make noisy changes that obscure the real differences.

Committers should not be afraid to add changes that still need attention. This enables prompt patch application and eases the load on the individual committer. An interesting side-effect is that it encourages community growth. See this discussion and its previous threads for an excellent view regarding this topic.

8. Contribution and acknowledgement

Some principles of open development at ASF are to ensure that each contributor is recognised and feels a productive part of the community, and to encourage diversity, respect, and equality. Key to this is the recognition of contributions from individuals in a manner that also recognises the community

effort that made it all possible. It is important to remember that there is no concept of individual leadership. See the discussion of meritocracy and other sections of the How the ASF works document.

In an Open Source Project, or more importantly, a project developed using an open process, such as Apache Forrest, most contributions of actual code are supported by, or at least *should* be supported by, design discussion, oversight, testing, documentation, bug fixes and much more. No code contribution is an independent unit of work (or should not be). It is therefore impossible to credit individual contributors, it is simply unmanageable, even if it is possible to identify each part of a contribution.

At Apache Forrest we use the following method to provide recognition:

- All developers encourage other developers to participate on the mailing lists, treat each other with respect, and openly collaborate. This enables the contributors to feel a part of the project and shows that their discussion and ideas are valuable. These replies enhance the presence of their name in the email archives and search engines.
- Encourage contributors to add patches via the issue tracker. This also enables clear tracking of the issue and by default specifically shows who was the contributor.
- When committers apply the patch, they refer to the issue number and the contributor's name. This enables linkage between the issue tracker and the Subversion history. It adds the contributor's name to the mail archives.
- Committers apply patches as soon as possible. This keeps the contributor enthused and shows them that their work is valuable.
- Committers add an entry for each significant contribution to the top-level changes document (site-author/status.xml) and detailed entries to the relevant plugin's changes document. This enables linkage to the relevant issue and shows the contributor's name. It also shows the initials of the committer who did the work to add the patch.
- When committers are adding their own work, they similarly add entries to the "changes" documents. Their initials are added to the entry.
- The existing PMC will notice new contributors who are committed. It eventually invites them to become new committers/PMC members. See the notes about this topic.
- Committers/PMC members are listed.

As discussed above, there is no specific documentation which lists each contributor and their work. For those who are interested there are various mechanisms: Use general internet search services; use search services provided by various third-party mail archives; search the "svn" mailing list using committer IDs and using contributor names; browse the changes page; use 'svn log' and 'svn blame'.

9. Development procedure

Note:

This section is still under development. The issues have been discussed on the mailing list. Decisions need to be put into words, so that we do not need to revisit the topic.

Development work is done on the trunk of SVN. Wherever possible, functionality is contained in "plugins". There are "release branches" of SVN, e.g. forrest_07_branch.

FIXME (open):

The following issues still need to be added above. There are also some relevant email threads, from which we need to extract info.

```
* Define our policy for adding changes to release branch.
```

```
* Define the purpose of the "whiteboard/incubator".
* Declare that we only really maintain the current release branch
  (although contributed patches will be applied).
* When to create a temporary branch for development and when/how to merge.


Some of the many relevant threads in no particular order ...

http://marc.theaimsgroup.com/?t=113344003500003
Whiteboard usage - rename it to incubator

http://marc.theaimsgroup.com/?t=112798856400001
Starting a 1.0 development  (Re: [Proposal] rollback)

http://marc.theaimsgroup.com/?l=forrest-dev&m=111968323717620
http://marc.theaimsgroup.com/?l=forrest-dev&m=111983663526246
http://marc.theaimsgroup.com/?t=111970529900001
Project participation and hackability    (was: [VOTE] merge locationmap

http://marc.theaimsgroup.com/?t=112507381300001
use of whiteboard in forrest

http://marc.theaimsgroup.com/?t=112504310100005
[Proposal] Development process and a stable trunk

http://marc.theaimsgroup.com/?l=forrest-dev&m=113521408020541
when to make a release of a branch
http://marc.theaimsgroup.com/?l=forrest-dev&m=112643002807899
How to apply an update to 0.7

http://marc.theaimsgroup.com/?t=113616009300002
[RT] "Last known working snapshot" of Forrest head

http://marc.theaimsgroup.com/?t=113830245600001
[Proposal] code freeze on dispatcher related resources

http://marc.theaimsgroup.com/?t=114667400800004
[Proposal] Refining our Development Process

Document our use of Branches for development
http://issues.apache.org/jira/browse/FOR-631

http://marc.info/?t=117858638000084
Re: fixing bugs in trunk or branch
and see prior discussion in response to r535593
```