Rocky Enterprise Linux 9.2 Manual Pages on command 'perlpolicy.1'

$ man perlpolicy.1

NAME

perlpolicy - Various and sundry policies and commitments related to the Perl core

DESCRIPTION

This document is the master document which records all written policies about how the Perl 5 Porters collectively develop and maintain the Perl core.

GOVERNANCE

Perl 5 Porters

Subscribers to perl5-porters (the porters themselves) come in several flavours. Some are quiet curious lurkers, who rarely pitch in and instead watch the ongoing development to ensure they're forewarned of new changes or features in Perl. Some are representatives of vendors, who are there to make sure that Perl continues to compile and work on their platforms. Some patch any reported bug that they know how to fix, some are actively patching their pet area (threads, Win32, the regexp -engine), while others seem to do nothing but complain. In other words, it's your usual mix of technical people.

Among these people are the core Perl team. These are trusted volunteers involved in the ongoing development of the Perl language and interpreter. They are not required to be language developers or committers.

Over this group of porters presides Larry Wall. He has the final word in what does and does not change in any of the Perl programming languages. These days, Larry spends most of his time on Raku, while Perl 5 is shepherded by a steering council of porters responsible for deciding what goes into each release and ensuring that releases happen on a regular basis.

Larry sees Perl development along the lines of the US government: there's the Legislature (the porters, represented by the core team), the Executive branch (the steering council), and the Supreme Court (Larry). The legislature can discuss and submit patches to the executive branch all they like, but the executive branch is free to veto them. Rarely, the Supreme Court will side with the executive branch over the legislature, or the legislature over the executive branch. Mostly, however, the legislature and the executive branch are supposed to get along and work out their differences without impeachment or court cases.

You might sometimes see reference to Rule 1 and Rule 2. Larry's power as Supreme Court is expressed in The Rules:

1. Larry is always by definition right about how Perl should behave. This means he has final veto power on the core functionality.
2. Larry is allowed to change his mind about any matter at a later date, regardless of whether he previously invoked Rule 1.

Got that? Larry is always right, even when he was wrong. It's rare to see either Rule exercised, but they are often alluded to.

For the specifics on how the members of the core team and steering council are elected or rotated, consult perlgov, which spells it all out in detail.

MAINTENANCE AND SUPPORT

Perl 5 is developed by a community, not a corporate entity. Every change contributed to the Perl core is the result of a donation. Typically, these donations are contributions of code or time by individual members of our community. On occasion, these donations come in the form of corporate or organizational sponsorship of a particular individual or project.

As a volunteer organization, the commitments we make are heavily dependent on the goodwill and hard work of individuals who have no obligation to contribute to Perl.

That being said, we value Perl's stability and security and have long had an unwritten covenant with the broader Perl community to support and maintain releases of Perl.

This document codifies the support and maintenance commitments that the Perl community should expect from Perl's developers:

? We "officially" support the two most recent stable release series. 5.26.x and earlier are now out of support. As of the release of 5.32.0, we will "officially" end support for Perl 5.28.x, other than providing security updates as described below.

? To the best of our ability, we will attempt to fix critical issues in the two most

recent stable 5.x release series. Fixes for the current release series take
precedence over fixes for the previous release series.

? To the best of our ability, we will provide "critical" security patches / releases for any major version of Perl whose 5.x.0 release was within the past three years. We can only commit to providing these for the most recent .y release in any 5.x.y series.

? We will not provide security updates or bug fixes for development releases of Perl.

? We encourage vendors to ship the most recent supported release of Perl at the time of their code freeze.

? As a vendor, you may have a requirement to backport security fixes beyond our 3 year support commitment. We can provide limited support and advice to you as you do so and, where possible will try to apply those patches to the relevant -maint branches in git, though we may or may not choose to make numbered releases or "official" patches available. See "SECURITY VULNERABILITY CONTACT INFORMATION" in perlsec for details on how to begin that process.

BACKWARD COMPATIBILITY AND DEPRECATION

Our community has a long-held belief that backward-compatibility is a virtue, even when the functionality in question is a design flaw.

We would all love to unmake some mistakes we've made over the past decades. Living with every design error we've ever made can lead to painful stagnation. Unwinding our mistakes is very, very difficult. Doing so without actively harming our users is nearly impossible.

Lately, ignoring or actively opposing compatibility with earlier versions of Perl has come into vogue. Sometimes, a change is proposed which wants to usurp syntax which previously had another meaning. Sometimes, a change wants to improve previously-crazy semantics.

Down this road lies madness.

Requiring end-user programmers to change just a few language constructs, even language constructs which no well-educated developer would ever intentionally use is tantamount to saying "you should not upgrade to a new release of Perl unless you have 100% test coverage and can do a full manual audit of your codebase." If we were to have tools capable of reliably upgrading Perl source code from one version of Perl to another, this concern could be significantly mitigated.

We want to ensure that Perl continues to grow and flourish in the coming years and decades, but not at the expense of our user community.

Existing syntax and semantics should only be marked for destruction in very limited circumstances. If they are believed to be very rarely used, stand in the way of actual improvement to the Perl language or perl interpreter, and if affected code can be easily updated to continue working, they may be considered for removal. When in doubt, caution dictates that we will favor backward compatibility. When a feature is deprecated, a statement of reasoning describing the decision process will be posted, and a link to it will be provided in the relevant perldelta documents.

Using a lexical pragma to enable or disable legacy behavior should be considered when appropriate, and in the absence of any pragma legacy behavior should be enabled. Which backward-incompatible changes are controlled implicitly by a 'use v5.x.y' is a decision which should be made by the steering council in consultation with the community.

Historically, we've held ourselves to a far higher standard than backward-compatibility -- bugward-compatibility. Any accident of implementation or unintentional side-effect of running some bit of code has been considered to be a feature of the language to be defended with the same zeal as any other feature or functionality. No matter how frustrating these unintentional features may be to us as we continue to improve Perl, these unintentional features often deserve our protection. It is very important that existing software written in Perl continue to work correctly. If end-user developers have adopted a bug as a feature, we need to treat it as such.

New syntax and semantics which don't break existing language constructs and syntax have a much lower bar. They merely need to prove themselves to be useful, elegant, well designed, and well tested. In most cases, these additions will be marked as experimental for some time. See below for more on that.

Terminology

To make sure we're talking about the same thing when we discuss the removal of features or functionality from the Perl core, we have specific definitions for a few words and phrases.

experimental

If something in the Perl core is marked as experimental, we may change its behaviour, deprecate or remove it without notice. While we'll always do our best to smooth the transition path for users of experimental features, you should contact the perl5-porters mailinglist if you find an experimental feature useful and want to help shape its future.

Experimental features must be experimental in two stable releases before being marked non-experimental. Experimental features will only have their experimental status revoked when they no longer have any design-changing bugs open against them and when they have remained unchanged in behavior for the entire length of a development cycle. In other words, a feature present in v5.20.0 may be marked no longer experimental in v5.22.0 if and only if its behavior is unchanged throughout all of v5.21.

deprecated

If something in the Perl core is marked as deprecated, we may remove it from the core in the future, though we might not. Generally, backward incompatible changes will have deprecation warnings for two release cycles before being removed, but may be removed after just one cycle if the risk seems quite low or the benefits quite high.

As of Perl 5.12, deprecated features and modules warn the user as they're used. When a module is deprecated, it will also be made available on CPAN. Installing it from CPAN will silence deprecation warnings for that module.

If you use a deprecated feature or module and believe that its removal from the Perl core would be a mistake, please contact the perl5-porters mailinglist and plead your case. We don't deprecate things without a good reason, but sometimes there's a counterargument we haven't considered. Historically, we did not distinguish between "deprecated" and "discouraged" features.

discouraged

From time to time, we may mark language constructs and features which we consider to have been mistakes as discouraged. Discouraged features aren't currently candidates for removal, but we may later deprecate them if they're found to stand in the way of a significant improvement to the Perl core.

removed

Once a feature, construct or module has been marked as deprecated, we may remove it from the Perl core. Unsurprisingly, we say we've removed these things. When a module is removed, it will no longer ship with Perl, but will continue to be available on CPAN.

MAINTENANCE BRANCHES

New releases of maintenance branches should only contain changes that fall into one of the "acceptable" categories set out below, but must not contain any changes that fall into one of the "unacceptable" categories. (For example, a fix for a crashing bug must not be included if it breaks binary compatibility.)

It is not necessary to include every change meeting these criteria, and in general the focus should be on addressing security issues, crashing bugs, regressions and serious installation issues. The temptation to include a plethora of minor changes that don't affect the installation or execution of perl (e.g. spelling corrections in documentation) should be resisted in order to reduce the overall risk of overlooking something. The intention is to create maintenance releases which are both worthwhile and which users can have full confidence in the stability of. (A secondary concern is to avoid burning out the maint-release manager or overwhelming other committers voting on changes to be included (see "Getting changes into a maint branch" below).)

The following types of change may be considered acceptable, as long as they do not also fall into any of the "unacceptable" categories set out below:

? Patches that fix CVEs or security issues. These changes should be passed using the security reporting mechanism rather than applied directly; see "SECURITY VULNERABILITY CONTACT INFORMATION" in perlsec.

? Patches that fix crashing bugs, assertion failures and memory corruption but which do not otherwise change perl's functionality or negatively impact performance.

? Patches that fix regressions in perl's behavior relative to previous releases, no matter how old the regression, since some people may upgrade from very old versions of perl to the latest version.

? Patches that fix bugs in features that were new in the corresponding 5.x.0 stable release.

? Patches that fix anything which prevents or seriously impacts the build or installation of perl.

? Portability fixes, such as changes to Configure and the files in the hints/ folder.

? Minimal patches that fix platform-specific test failures.

? Documentation updates that correct factual errors, explain significant bugs or deficiencies in the current implementation, or fix broken markup.

? Updates to dual-life modules should consist of minimal patches to fix crashing bugs or security issues (as above). Any changes made to dual-life modules for which CPAN is canonical should be coordinated with the upstream author.

The following types of change are NOT acceptable:

? Patches that break binary compatibility. (Please talk to the steering council.)

? Patches that add or remove features.

? Patches that add new warnings or errors or deprecate features.

? Ports of Perl to a new platform, architecture or OS release that involve changes to the implementation.

? New versions of dual-life modules should NOT be imported into maint. Those belong in the next stable series.

If there is any question about whether a given patch might merit inclusion in a maint release, then it almost certainly should not be included.

Getting changes into a maint branch

Historically, only the single-person project manager cherry-picked changes from bleadperl into maintperl. This has scaling problems. At the same time, maintenance branches of stable versions of Perl need to be treated with great care. To that end, as of Perl 5.12, we have a new process for maint branches.

Any committer may cherry-pick any commit from blead to a maint branch by first adding an entry to the relevant voting file in the maint-votes branch announcing the commit as a candidate for back-porting, and then waiting for at least two other committers to add their votes in support of this (i.e. a total of at least three votes is required before a commit may be back-ported).

Most of the work involved in both rounding up a suitable set of candidate commits and cherry-picking those for which three votes have been cast will be done by the maint branch release manager, but anyone else is free to add other proposals if they're keen to ensure certain fixes don't get overlooked or fear they already have been.

Other voting mechanisms may also be used instead (e.g. sending mail to perl5-porters and at least two other committers responding to the list giving their assent), as long as the same number of votes is gathered in a transparent manner. Specifically, proposals of which changes to cherry-pick must be visible to everyone on perl5-porters so that the views of everyone interested may be heard.

It is not necessary for voting to be held on cherry-picking perldelta entries associated with changes that have already been cherry-picked, nor for the maint-release manager to obtain votes on changes required by the Porting/release_managers_guide.pod where such changes can be applied by the means of cherry-picking from blead.

CONTRIBUTED MODULES

A Social Contract about Artistic Control

What follows is a statement about artistic control, defined as the ability of authors of packages to guide the future of their code and maintain control over their work. It is a recognition that authors should have control over their work, and that it is a responsibility of the rest of the Perl community to ensure that they retain this control.
It is an attempt to document the standards to which we, as Perl developers, intend to hold ourselves. It is an attempt to write down rough guidelines about the respect we owe each other as Perl developers.
This statement is not a legal contract. This statement is not a legal document in any way, shape, or form. Perl is distributed under the GNU Public License and under the Artistic License; those are the precise legal terms. This statement isn't about the law or licenses. It's about community, mutual respect, trust, and good-faith cooperation.
We recognize that the Perl core, defined as the software distributed with the heart of Perl itself, is a joint project on the part of all of us. From time to time, a script, module, or set of modules (hereafter referred to simply as a "module") will prove so widely useful and/or so integral to the correct functioning of Perl itself that it should be distributed with the Perl core. This should never be done without the author's explicit consent, and a clear recognition on all parts that this means the module is being distributed under the same terms as Perl itself. A module author should realize that inclusion of a module into the Perl core will necessarily mean some loss of control over it, since changes may occasionally have to be made on short notice or for consistency with the rest of Perl.
Once a module has been included in the Perl core, however, everyone involved in maintaining Perl should be aware that the module is still the property of the original author unless the original author explicitly gives up their ownership of it. In particular:

? The version of the module in the Perl core should still be considered the work of the original author. All patches, bug reports, and so forth should be fed back to them. Their development directions should be respected whenever possible.

? Patches may be applied by the steering council without the explicit cooperation of the module author if and only if they are very minor, time-critical in some fashion (such as urgent security fixes), or if the module author cannot be reached. Those patches must still be given back to the author when possible, and if the author decides on an alternate fix in their version, that fix should be strongly preferred unless there is a serious problem with it. Any changes not endorsed by the author should be marked as such, and the contributor of the change acknowledged.

? The version of the module distributed with Perl should, whenever possible, be the latest version of the module as distributed by the author (the latest non-beta version in the case of public Perl releases), although the steering council may hold off on upgrading the version of the module distributed with Perl to the latest version until the latest version has had sufficient testing.

In other words, the author of a module should be considered to have final say on modifications to their module whenever possible (bearing in mind that it's expected that everyone involved will work together and arrive at reasonable compromises when there are disagreements).

As a last resort, however:

If the author's vision of the future of their module is sufficiently different from the vision of the steering council and perl5-porters as a whole so as to cause serious problems for Perl, the steering council may choose to formally fork the version of the module in the Perl core from the one maintained by the author. This should not be done lightly and should always if at all possible be done only after direct input from Larry.

If this is done, it must then be made explicit in the module as distributed with the Perl core that it is a forked version and that while it is based on the original author's work, it is no longer maintained by them. This must be noted in both the documentation and in the comments in the source of the module.

Again, this should be a last resort only. Ideally, this should never happen, and every possible effort at cooperation and compromise should be made before doing this. If it does prove necessary to fork a module for the overall health of Perl, proper credit must be given to the original author in perpetuity and the decision should be constantly re-evaluated to see if a remerging of the two branches is possible down the road.

In all dealings with contributed modules, everyone maintaining Perl should keep in mind that the code belongs to the original author, that they may not be on perl5-porters at any given time, and that a patch is not official unless it has been integrated into the author's copy of the module. To aid with this, and with points #1, #2, and #3 above, contact information for the authors of all contributed modules should be kept with the Perl distribution.

Finally, the Perl community as a whole recognizes that respect for ownership of code, respect for artistic control, proper credit, and active effort to prevent unintentional code skew or communication gaps is vital to the health of the community and Perl itself. Members of a community should not normally have to resort to rules and laws to deal with each other, and this document, although it contains rules so as to be clear, is about an attitude and general approach. The first step in any dispute should be open communication, respect for opposing views, and an attempt at a compromise. In nearly every circumstance nothing more will be necessary, and certainly no more drastic measure should be used until every avenue of communication and discussion has failed.

DOCUMENTATION

Perl's documentation is an important resource for our users. It's incredibly important for Perl's documentation to be reasonably coherent and to accurately reflect the current implementation.

Just as P5P collectively maintains the codebase, we collectively maintain the documentation. Writing a particular bit of documentation doesn't give an author control of the future of that documentation. At the same time, just as source code changes should match the style of their surrounding blocks, so should documentation changes.

Examples in documentation should be illustrative of the concept they're explaining. Sometimes, the best way to show how a language feature works is with a small program the reader can run without modification. More often, examples will consist of a snippet of code containing only the "important" bits. The definition of "important" varies from snippet to snippet. Sometimes it's important to declare "use strict" and "use warnings", initialize all variables and fully catch every error condition. More often than not, though, those things obscure the lesson the example was intended to teach.

As Perl is developed by a global team of volunteers, our documentation often contains spellings which look funny to somebody. Choice of American/British/Other spellings is left as an exercise for the author of each bit of documentation. When patching documentation, try to emulate the documentation around you, rather than changing the existing prose.

In general, documentation should describe what Perl does "now" rather than what it used to do. It's perfectly reasonable to include notes in documentation about how behaviour has changed from previous releases, but, with very few exceptions, documentation isn't "dual-life" -- it doesn't need to fully describe how all old versions used to work.

The official forum for the development of perl is the perl5-porters mailing list, mentioned above, and its bugtracker at GitHub. Posting to the list and the bugtracker is not a right: all participants in discussion are expected to adhere to a standard of conduct.

? Always be civil.

? Heed the moderators.

Civility is simple: stick to the facts while avoiding demeaning remarks, belittling other individuals, sarcasm, or a presumption of bad faith. It is not enough to be factual. You must also be civil. Responding in kind to incivility is not acceptable. If you relay otherwise-unposted comments to the list from a third party, you take responsibility for the content of those comments, and you must therefore ensure that they are civil.

While civility is required, kindness is encouraged; if you have any doubt about whether you are being civil, simply ask yourself, "Am I being kind?" and aspire to that.

If the list moderators tell you that you are not being civil, carefully consider how your words have appeared before responding in any way. Were they kind? You may protest, but repeated protest in the face of a repeatedly reaffirmed decision is not acceptable.

Repeatedly protesting about the moderators' decisions regarding a third party is also unacceptable, as is continuing to initiate off-list contact with the moderators about their decisions.

Unacceptable behavior will result in a public and clearly identified warning. A second instance of unacceptable behavior from the same individual will result in removal from the mailing list and GitHub issue tracker, for a period of one calendar month. The rationale for this is to provide an opportunity for the person to change the way they act.

After the time-limited ban has been lifted, a third instance of unacceptable behavior will result in a further public warning. A fourth or subsequent instance will result in an indefinite ban. The rationale is that, in the face of an apparent refusal to change behavior, we must protect other community members from future unacceptable actions. The moderators may choose to lift an indefinite ban if the person in question affirms they will not transgress again.

Removals, like warnings, are public.

The list of moderators will be public knowledge. At present, it is: Karen Etheridge, Neil Bowers, Nicholas Clark, Ricardo Signes, Todd Rinaldo.

"Social Contract about Contributed Modules" originally by Russ Allbery <rra@stanford.edu>

and the perl5-porters.