

The Community's Rewrite of Perl

Perl 6 Essentials



O'REILLY[®]

*Allison Randal,
Dan Sugalski & Leopold Tötsch*

Perl 6 Essentials

Other Perl Resources from O'Reilly

Related titles	Programming Perl	Mastering Regular
	Learning Perl	Expressions
	Perl Cookbook	Perl for System
	CGI Programming with	Administration
	Perl	Programming Web
	Computer Science & Perl	Services with Perl
	Programming	Perl Pocket Reference
	Embedding Perl in HTML	Perl in a Nutshell
	with Mason	Perl Graphics
		Programming

Perl Books Resource Center

perl.oreilly.com is a complete catalog of O'Reilly's books on Perl and related technologies, including sample chapters and code examples.

Perl.com is the central web site for the Perl community. It is the perfect starting place for finding out everything there is to know about Perl.

Conferences

O'Reilly & Associates brings diverse innovators together to nurture the ideas that spark revolutionary industries. We specialize in documenting the latest tools and systems, translating the innovator's knowledge into useful skills for those in the trenches. Visit *conferences.oreilly.com* for our upcoming events.



Safari Bookshelf (*safari.oreilly.com*) is the premier online reference library for programmers and IT professionals. Conduct searches across more than 1,000 books. Subscribers can zero in on answers to time-critical questions in a matter of seconds. Read the books on your Bookshelf from cover to cover or simply flip to the page you need. Try it today with a free trial.

Perl 6 Essentials

*Allison Randal, Dan Sugalski,
and Leopold Tötsch*

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

Project Overview

Conceptual integrity in turn dictates that the design must proceed from one mind, or from a very small number of agreeing resonant minds.

—Frederick Brooks Jr.
The Mythical Man Month

Perl 6 is the next major version of Perl. It’s a complete rewrite of the interpreter, and a significant update of the language itself. The goal of Perl 6 is to add support for much-needed new features, and still be cleaner, faster, and easier to use.

The Perl 6 project is vast and complex, but it isn’t complicated. The project runs on a simple structure with very little management overhead. That’s really the only way it could run. The project doesn’t have huge cash or time resources. Its only resource is the people who believe in the project enough to spend their off-hours—their “relaxation” time—working to see it completed. This chapter is as much about people as it is about Perl.

The Birth of Perl 6

Back on July 18, 2000, the second day of the fourth Perl Conference (TPC 4), a small band of Perl geeks gathered to prepare for a meeting of the Perl 5 Porters later that day. The topic at hand was the current state of the Perl community. Four months had passed since the 5.6.0 release of Perl, and although it introduced some important features, none were revolutionary.

There had been very little forward movement in the previous year. It was generally acknowledged that the Perl 5 codebase had grown difficult to maintain. At the same time, infighting on the *perl5-porters* list had grown so intense that some of the best developers decided to leave. It was time for a

change, but no one was quite sure what to do. They started conservatively with plans to change the organization of Perl development.

An hour into the discussion, around the time most people nod off in any meeting, Jon Orwant (the reserved, universally respected editor of the Perl Journal) stepped quietly into the room and snapped everyone to attention with an entirely uncharacteristic and well-planned gesture. *Smash!* A coffee mug hit the wall. “We are *@\$!-ed (*Crash!*) unless we can come up with something that will excite the community (*Pow!*), because everyone’s getting bored and going off and doing other things! (*Bam!*)” (At least, that’s basically how Larry tells it. As is usually the case with events like this, no one remembers exactly what Jon said.)

Awakened by this display, the group started to search for a real solution. The language needed room to grow. It needed the freedom to evaluate new features without the obscuring weight of legacy code. The community needed something to believe in, something to get excited about.

Within a few hours the group settled on Perl 6, a complete rewrite of Perl. The plan wasn’t just a language change, just an implementation change, or just a social change. It was a paradigm shift. Perl 6 would be the community’s rewrite of Perl, and the community’s rewrite of itself.

Would Perl 6, particularly Perl 6 as a complete rewrite, have happened without this meeting? Almost certainly. The signs appeared on the lists, in conferences, and in journals months in advance. If it hadn’t started that day, it would have happened a week later, or perhaps a few months later, but it would have happened. It was a step the community needed to take.

In the Beginning...

Let’s pause and consider Perl development up to that fateful meeting. Perl 6 is just another link in the chain. The motivations behind it and the directions it will take are partially guided by history.

Perl was first developed in 1987 by Larry Wall while he was working as a programmer for Unisys. After creating a configuration and monitoring system for a network that spanned the two American coasts, he was faced with the task of assembling usable reports from log files scattered across the network. The available tools simply weren’t up to the job. A linguist at heart, Larry set out to create his own programming language, which he called *perl*. He released the first version of Perl on December 18, 1987. He made it freely available on Usenet (this was before the Internet took over the world, remember), and quickly a community of Perl programmers grew.

The early adopters of Perl were system administrators who had hit the wall with shell scripting, *awk*, and *sed*. However, in the mid-1990s Perl's audience exploded with the advent of the Web, as Perl was tailor-made for CGI scripting and other web-related programming.

Meantime, the Perl language itself kept growing, as Larry and others kept adding new features. Probably the most revolutionary change in Perl (until Perl 6, of course) was the addition of packages, modules, and object-oriented programming with Perl 5. While this made the transition period from Perl 4 to Perl 5 unusually long, it breathed new life into the language by providing a modern, modular interface. Before Perl 5, Perl was considered simply a scripting language; after Perl 5, it was considered a full-fledged programming language.

Larry, meanwhile, started taking a back seat to Perl development and allowed others to take responsibility for adding new features and fixing bugs in Perl. The Perl 5 Porters (p5p) mailing list became the central clearing-house for bug reports or proposed changes to the Perl language, with the “pumpkin holder” (also known as the “pumpking”) being the programmer responsible for implementing the patches and distributing them to the rest of the list for review. Larry continued to follow Perl development, but like a parent determined not to smother his children, he stayed out of the day-to-day development, limiting his involvement to situations in which he was truly needed.

Although you might think that the birth of the Perl 6 project would be the first nail in the coffin for Perl 5, that's far from the case. If anything, Perl 5 has had a huge resurgence of development, with Perl 5.7.0 released only two weeks after the initial decision to go ahead with Perl 6. Perl 5.8, spearheaded by Jarkko Hietaniemi and released in July 2002, includes usable Unicode support, a working threads interface, safe signals, and a significant improvement of the internals with code cleanup, bug fixes, better documentation, and more than quadrupled test coverage. Hugo van der Sanden is the pumpking for 5.9–5.10. Plans for those releases include enhancements to the regular expression engine, further internals cleanup and a “use perl6ish” pragma that will integrate many of the features of Perl 6. Perl 5 is active and thriving, and will continue to be so even after the release of Perl 6.0.

The Continuing Mission

Much has changed since the early days of the project. New people join the group and others leave in a regular “changing of the guard” pattern. Plans change as the work progresses, and the demands of the work and the needs

of the community become clearer. Today the Perl 6 project has three major parts: language design, internals, and documentation. Each branch is relatively autonomous, though there is a healthy amount of coordination between them.

Language Design

As with all things Perl, the central command of the language design process is Larry Wall, the creator of the Perl language. Larry is supported by the rest of the design team: Damian Conway, Allison Randal, Dan Sugalski, Hugo van der Sanden, and chromatic. We speak in weekly teleconferences and also meet face-to-face a few times a year to hash out ideas for the design documents, or to work through roadblocks standing in the way of design or implementation. The group is diverse, including programmers-for-hire, Perl trainers, and linguists with a broad spectrum of interests and experiences. This diversity has proved quite valuable in the design process, as each member is able to see problems in the design or potential solutions that the other members missed.

Requests for comments (RFCs)

The first step in designing the new language was the RFC (Request For Comments) process. This spurred an initial burst of community involvement. Anyone was free to submit an RFC on any subject, whether it was as small as adding an operator, or as big as reworking OO syntax. Most of the proposals were really quite conservative. The RFCs followed a standard format so they would be easier to read and easier to compare.

Each RFC was subject to peer review, carried out in an intense few weeks around October 2000. One thing the RFC process demonstrated was that the Perl community still wasn't quite ready to move beyond the infighting that had characterized Perl 5 Porters earlier that year.*

Even though few RFCs have been accepted without modification, the process identified a large number of irritants in the language. These have served as signposts for later design efforts.

* Mark-Jason Dominus wrote an excellent critique of the RFC process (<http://www.perl.com/pub/a/2000/11/perl6rfc.html>). It may seem harsh to people accustomed to the more open and tolerant community of today, but it's an accurate representation of the time when it was written.

Apocalypses and Exegeses

The Apocalypses* and Exegeses† are an important part of the design process. Larry started the Apocalypse series as a systematic way of answering the RFCs. Each Apocalypse corresponds to a chapter in his book *Programming Perl*, and addresses the features in the chapter that are likely to change.

However, the Apocalypses have become much more than a simple response to RFCs. Larry has a startling knack for looking at 12 solutions to a problem, pulling out the good bits from each one, and combining them into a solution that is 10 times better than any of the proposals alone. The Apocalypses are an excellent example of this “Larry Effect.” He addresses each relevant RFC, and gives reasons why he accepted or rejected various pieces of it. But each Apocalypse also goes beyond a simple “yes” and “no” response to attack the roots of the problems identified in the RFCs.

Damian Conway’s Exegeses are extensions of each Apocalypse. Each Exegesis is built around a practical code example that applies and explains the new ideas.

The p6l mailing list

The next body of design work is the Perl 6 Language mailing list (*perl6-language@perl.org*), often fondly referred to as “p6l.” Luke Palmer has been deputized as unofficial referee of the list. He answers questions that don’t require the direct involvement of the design team or that have been answered before. He also keeps an eye out for good suggestions to make sure the design team doesn’t miss them in the sea of messages. The list has approximately 40 regular contributors in any given month, as well as a large number of occasional posters and lurkers. Some people have participated since the very beginning; others appear for a few months and move on.

Even though the individuals change, the general tone of p6l is the same. It’s an open forum for any ideas on the user-visible parts of Perl 6. In the typical pattern, one person posts an idea and 5 to 10 people respond with criticisms or suggestions. The list periodically travels down a speculative thread like a runaway train, but these eventually run out of steam. Then Larry picks out the golden bits and gently tells the rest that no, he never intended Perl 6 to have hyper-vulcan mechanoid scooby-dooby-doo. Even when Larry doesn’t post, he follows the list and the traffic serves as a valuable catalyst for his thoughts.

* An “apocalypse” in the sense of “revelation,” not “end of the world.”

† An “exegesis” is an explanation or interpretation of a text.

Internals

Parrot is a grandiose idea that turned out to be more realistic than anyone originally could have believed: why not have a single interpreter for several languages? Unlike the parent Perl 6 project, which was launched in a single day, the plan for Parrot formed in bits and pieces over the space of a year.

On April 1, 2001, Simon Cozens published an article titled “Programming Parrot” as an April Fools’ joke (<http://www.perl.com/pub/a/2001/04/01/parrot.htm>). It was a contrived interview with Larry Wall and Guido van Rossum detailing their plans to merge Python and Perl into a new language called Parrot. A few months later, when Perl 6 internals began to take an independent path within the larger project, they dubbed the subproject “Parrot” in a fitting turn of life imitating art.

Early Steps Toward Perl 6 Internals

The earliest progress toward implementing Perl 6 started before the current incarnation of Perl 6 was even conceived. The Topaz project, started in 1998, was spearheaded by Chip Salzenberg. It was a reimplement of Perl 5 written in C++. The project was abandoned, but many of the goals and intended features for Topaz were adopted for Perl 6 internals, and the difficulties Topaz encountered were also valuable guides.

Sapphire was another early prototype that influenced the shape of Perl 6 internals. It was a one-week project in September 2000. The brainchild of Simon Cozens, Sapphire was another rewrite of Perl 5 internals. It was never intended for release, only as an experiment to see how far the idea could go in a week, and what lessons could be learned.

The plan for Parrot was to build a language-neutral runtime environment. It would support all the features of dynamic languages such as Python, Ruby, Scheme, Befunge, and others. It would have threading and Unicode support (two of the most problematic features to add into Perl 5 code) built in from the start. It would support exceptions and compilation to bytecode, and have clean extension and embedding mechanisms.

The language-neutral interpreter was originally just a side effect of good design. Keeping the implementation independent of the syntax would make the code cleaner and easier to maintain. One practical advantage of this design was that Parrot development could begin even though the Perl 6 language specification was still in flux.

The bigger win in the long term, though, was that since Parrot would support the features of the major dynamic languages and wasn't biased to a particular syntax, it could run all these languages with little additional effort. It's generally acknowledged that different languages are suited to different tasks. Picking which language will be used in a large software project is a common planning problem. There's never a perfect fit. It usually boils down to picking the language with the most advantages and the least noticeable disadvantages. The ability to combine multiple languages within a project could be a huge benefit. Use well-tested libraries from one language for one task. Take advantage of a clean way of expressing a particular problem domain in a second, without being forced to use it in areas where it's weak.

The modular design also benefits future language designers. Instead of targeting *lex/yacc* and reimplementing low-level features such as garbage collection and dynamic types, designers can write a parser that targets the Parrot virtual machine.

The internals development for Perl 6 falls to the Parrot project. Dan Sugal-ski leads the project as internals designer, and Steve Fink is the current pumpking. The Parrot project is largely autonomous. Dan coordinates with the rest of the design team to ensure that Parrot will be able to support the semantics Perl 6 will require, but the language designers have very little input into the details of implementation. Parrot isn't developed solely for Perl, but Perl 6 is entirely dependent on Parrot—it is the only interpreter for Perl 6.

The core communication line for the Parrot project is the mailing list, *perl6-internals@perl.org*, otherwise known as “p6i.” It's a much more business-like list than p6l. Workflow in Parrot takes the form of submitted patches. Anyone is free to submit a patch, and contributors who consistently submit valuable patches over a long period of time are granted check-in access to the CVS repository.

Documentation

Though adequate documentation has been a goal from the very beginning, the Perl 6 documentation project is a relatively recent addition. It operates under the guidance of Michael Lazzaro. The stated goal of the documentation project is to systematically walk through each Apocalypse and produce fully specified documentation from it. The results of the project are eventually intended to be the documentation released with Perl 6.0.

The task of the documenters is a difficult one. The specification for Perl 6 is still in development and constantly shifting, so they're shooting at a moving

target. The process is immensely valuable though, as it helps to identify inconsistencies or problems in the design that the broad brushstrokes of the Apocalypses miss. Sometimes it is the documentation process that causes the shift in language specification, as identified problems lead to solutions and the solutions, in turn, trigger changes throughout the system.

Supporting Structure

Last, but not least, is the glue that holds the project together. The highest praise belongs to Ask Björn Hansen and Robert Spier, who manage the email, revision control, and bug-tracking systems, as well as the web pages for Perl 6 and Parrot. Without these systems, the project would grind to a screeching halt.

Nathan Torkington and Allison Randal share the load of project management. Nat tends to handle outside interfacing while Allison tends to handle the nuts and bolts of the project, but neither role is set in stone. As is typical of open source development projects, managing the Perl 6 project is quite different from managing a commercial project of the same size and complexity. There are no schedules, no deadlines, no hiring and firing, and no salaries, bonuses, or stock options. There are no employees or bosses; there is very little hierarchy whatsoever. Management in this context isn't about giving orders, it's about making sure everyone has what they need to keep moving forward.

In the end, it is the developers themselves who hold the project together. Each individual bears their own share of the responsibility for finding a task that suits their skills, coordinating with others to keep duplicated effort minimal, and making sure the job gets done.