Perl 6

Damian Conway

School of Computer Science and Software Engineering
Monash University
Australia

damian@conway.org

Summary

1. A Brief History of Perl (1.000 to 5.6.0)

- A brief recapitulation of the origins and development of today's Perl

-15,000,000,000

- Universe begins

-4,500,000,000

- Earth coalesces

-3,600,000,000

- Life evolves

-400,000

- H. Sapiens evolves

-50,000

- Spoken language evolves

-4,000

- Mesopotamians invent writing

-100

- Greeks build first recorded mechanical computational device

1830

- Charles Babbage commences work on Analytical Engine
- Ada, Lady Lovelace adapts punch cards for program storage

1938

- Konrad Zuse develops the first fully digital computer

1954

- September: Larry Wall born

1957

- FORTRAN released

1958

- Texas Instruments builds the first silicon chip
- ALGOL released

1960

- LISP released

1960

- COBOL released

1972

- C released

1986

- C++ released

1987

- 18 December: Perl 1.000 released
  - `$calars`, `@rrays`, `%ssociative_arrays`
  - subroutines
  - lists
  - `<FILE>` I/O
  - `while`, `for` (C-like), `if`
  - patterns and `=~`
  - formats
  - `chop`, `die`, `each`, `split`, `join`, `select`

1988

- 5 June: Perl 2.000 released
  - Spencerian regexes
  - local variables

- recursive subroutine calls
- arrays interpolate into lists
- `foreach`
- file inclusion via `do $file;`
- warnings (`-w`)
- `sort`

1989

- 18 October: Perl 3.000 released
  - GPL
  - binary data handling (`pack` and `unpack`)
  - pass-by-reference subroutine arguments
  - subroutine prefix (&)
  - `undef` introduced
  - `mkdir`, `rmdir`, `flock`, `readlink`, `warn`, `dbmopen`, `dbmclose`, `dump`, `reverse`, `defined`

1991

- 21 May: Perl 4.000 released

- Artistic License
- array in scalar context gives length
- `cmp` and `<=>`
- `caller`, `splice`, `qx//`,

1994

- 18 October: Perl 5.000 released
  - OO mechanism
  - perldoc and pod
  - lexical scoping
  - `::` package delimiter
  - `abs`, `chr`, `uc`, `ucfirst`, `lc`, `lcfirst`, `chomp`, `glob`
  - `use` statement
  - `=>` as synonym for comma
  - tied variables

1995

- 13 March: Perl 5.001 released

- closures
- `$SIG{__WARN__}` and `$SIG{__DIE__}`

1996

- 29 February: Perl 5.002 released
  - prototypes
  - operator overloading
- 10 October: Perl 5.003 released
  - bug fixes only

1997

- 5 May: Perl 5.004 released
  - `$coderef->(@args)` syntax
  - native `printf` and `sprintf`
  - `use $VERSION`
  - `UNIVERSAL::isa` and `UNIVERSAL::can`
  - `m//gc`

1998

- 22 July: Perl 5.005 released
  - complete tied arrays and handles
  - pseudo-hashes
  - threads
  - compiler
  - OO exception handling (`die $ref`)
- 19 August: Topaz project begins

2000

- March 28: Perl 5.6.0 released
  - Unicode support
  - lvalue subroutines,
  - weak references
  - lexically scoped warnings
  - `our` declarations
  - support for binary numbers.

2. A Brief History of Perl 6

The Birth

- On Tuesday, July 18, 2000 , some of the perl5-porters met during the 4th Perl Conference.

- Larry recalls what happened:

  - *"An interesting thing happened. We spent the first hour gabbing about all sorts of political and organizational issues of a fairly boring and mundane nature.*

  - *Partway through, Jon Orwant comes in, and stands there for a few minutes listening, and then he very calmly walks over to the coffee service table in the corner, and there were about 20 of us in the room, and he picks up a coffee mug and throws it against the other wall and he keeps throwing coffee mugs against the other wall, and he says "we are #*@%-ed unless we can come up with something that will excite the community, because everyone's getting bored and going off and doing other things".*

  - *And he was right....so that sort of galvanized the meeting. He said "I don't care what you do, but you gotta do something big." And then he went away.*

- *Don't misunderstand me. This was the most perfectly planned tantrum you have ever seen. If any of you know Jon, he likes control. This was a perfectly controlled tantrum. It was amazing to see. I was thinking, "Should I get up and throw mugs too?"*
- *Anyway, so we started talking after that and the idea popped up that maybe we oughta rewrite Perl. The idea that occurred to me then was that we had some new technology: Perl has a decompiler now. You can take a Perl 5 script and compile it down to the bytecode, and take that and compile it back to Perl 5 code. And if you can do that, why not compile it back to Perl 6 code?*
- *So later that day we also had an open meeting of the perl porters, with about 50 people there, and we started handing out tasks about how we would do the redesign. And the next day in my keynote I announced that we were rewriting Perl."*

The Reasons (besides not being #*@%-ed)

- Technical, political, and social reasons for developing Perl 6...

Technical reasons for developing Perl 6

- The current hacked nature of the Perl5 internals is hindering Perl development and keeping revolutionary new changes from Perl.
- Also provides a very high barrier to entry for people who are interested in contributing to Perl, fixing bugs, or writing XS.
- Perl6 as an opportunity to clean up the implementation of Perl while still keeping a language recognizable as Perl.
- Simplify Perl and reduce the complexity of the Perl implementation.
- Attempt to reduce the disk size of the interpreter, the amount of code necessary to implement it, and its overall memory footprint.

- Offers the possibility to re-examine Perl syntax. Perl6 could allow programmers to write Perl programs in multiple syntaxes, such as Python, JavaScript and Perl5 instead of the Perl6 syntax.

- Richer, structured documentation syntax. POD can be extended, cleaned up, or more widely adopted.

Political reasons for developing Perl 6

- Terse, incomprehensible or misunderstood feature requests lead to much discussion but little implementation.

- Majority of feature requests should be submitted in an RFC or white paper format that is more structured than email but not overly burdensome.

- Over time, produce a library of feature requests, justifications and discussions that will clarify why specific decisions were made.

- Version skew management: upgrading a module for one program may break other programs. Mechanisms for installing multiple versions of a shared library resource are well known and should be incorporated into Perl module installation.

- Perl's support for reusing component software and CORBA, COM, XPCOM should improve.

- The basic Perl package should include common database, HTML, XML, and other common features. The core Perl module library does not reflect how most users expect to use Perl out of the box.

- Many software developers like Java in part because Java programs can ship without source code. Adding capabilities to ship Perl programs in a binary format is an area Perl6 can address.

Social and Community reasons for developing Perl 6

- Perl5-porters model running out of steam (or perhaps victims).

- Need a series of smaller, more structured working groups that focus on indiviudal aspects of Perl:
  - chairperson ("pump-prince")
  - goals
  - charter
  - timeframe
  - deadline
- Give Perl6 community a different structure, culture
- Perl will remain an anything-goes, postmodern culture, but we need to help focus everyone in a more productive fashion
- Multiple mailing lists will be available to alert people using Perl6 about new features, experimental releases, stable release candidates and expected release schedules.
- Developers of CPAN modules may need to update their modules for Perl6. The development team will communicate the issues involved in upgrading modules so that CPAN is not left behind in the great push to Perl6.

- Backward compatibility has always been important: Perl user could take Perl5 programs, run it through a compiler backend to produce a Perl6 version.

The Announcement

- Released by O'Reilly & Associates on July 19, 2000.

- LARRY WALL ANNOUNCES BROAD VISION FOR PERL 6

- In his keynote address to the O'Reilly Second Open Source Convention, Larry Wall laid out his vision for the future development of Perl 6. Larry reports that "Perl 6 development has begun in earnest".

- The next version of Perl is a chance for the language developers to both rewrite the internals and externals of Perl based on their experience from developing Perl 5, and Chip Salzenberg1s work with Topaz. Larry promises that Perl 6 will be "better, stronger, faster" and that there will be a clear, clean migration path from Perl 5 to Perl 6. A preview release of should be available by next summer.

- Perl 6 will use a development model that draws from the lessons learned from perl5-porters and other large open source projects such as the Apache web server and the Linux operating system. Development topics will be assigned to working groups which will work under a central project manager. Nat Torkington is the interim project manager. Details will be made available in the Perl 6 section of www.perl.org.

What Larry said

- During his "State of the Onion" talk, July 19, 2000.
- *"One of the things we love about Perl is that it supports many different styles of programming. That's something we never want to lose with Perl....In fact, there are many features we want to conserve in Perl, but music is continually re-inventing itself and so is Perl culture.*
- *"...occasionally there does come a time when we have to think like revolutionaries. Yesterday, a bunch of us radicals decided that it was time to alter the course of human events."*
- *"...today, I'd like to announce to the world that the effort to write Perl 6 has begun in earnest."*
- *"...you eventually come to a point where you say, "This is a really neat gismo, but we can do something better. Do we continue to make small improvements in the current design or do we redesign the interface to let us do what we would really do down the road?"*

- *"And, if we do a redesign, can we keep everything people like about the old design while getting rid of all the things people don't like about the thing they have right now?"*

- *"Well, that's kind of the state Perl is in right now. We really, really like what we have. We like it a lot, but we can think of lots of ways we can do it better and the things we'd like to do better come in several categories."*

- *"First, the language itself could use some revision. There are many historical warts on Perl that wouldn't have been there if I'd known what I was doing...I'm more of a competent language designer than I was 13 years ago and I have a lot more help these days, plus it's time to steal all the good ideas we can from those other languages that developed in the last decade."*

- *"...we're actually in a much better position than when I designed Perl 5. Nowadays, we have code back-ins, such as B::B Parse, that can spit out the Perl code corresponding to the compiled syntax tree. If you think about that, it means that it would be relatively easy to make it spit out a closely related language, such as Perl 6."*

- *"...for the first time in history we have the opportunity to make some incompatible fixes to Perl while preserving a migration path for the current code."*

- *"Of course, we are not interested in breaking things just to break things, but I'm sure you can think of things you might have done differently. Myself, I really wish I'd made the system call return "true" on success rather than "false." I wish I'd made local time return the actual year and not the year minus 1900. I'd really love to throw out select file handle and there's general consensus that typeglobs may have outlived their usefulness."*

- *"My overriding goal for the redesign of Perl's language is that easy things should stay easy, hard things should get easier, and impossible things should get hard, as it were."*

- *"Another place we'd like to do better is in the implementation of the language...the internal APIs necessary to write extension modules could really use to be cleaned up."*

- *"So we've already started a redesign of Perl culture, trying to keep the good aspects and leaving behind the nonproductive aspects. We intend to abandon the Perl 5 porter's model of development, which demonstrably leads to a lot of talk but little action."*

- *"Instead we'll break down the design of Perl 6 and the maintenance of Perl 5 into manageable tasks given to meaningful working groups with meaningful charters and meaningful goals."*

- *"We are really jazzed about this. It is our belief that if Perl culture is designed right, Perl will be able to evolve into the language we need 20 years from now. It's also our belief that only a radical rethinking of both the Perl language and its implementation can energize the community in the long run"*

- *"We expect to have alpha code a year from now, for some definition of 'alpha'."*

- *"In the meantime, we are not abandoning Perl 5 anytime soon. We all like Perl 5 a lot. We all use it a lot. Many commercial interests will guarantee that Perl 5 continues to be well-maintained and stabilized for quite a few years to come, and we fully expect, given the history of Perl 4, that five years from now a lot of people will still be using Perl 5."*

- *"We have to be better, not just get there faster. Part of being better is making sure the stragglers don't get left behind. We are determined to do the right thing by everyone."*

The Reaction

- Concern

- *"Will Perl 5 source be 100% ok with Perl 6 or will I have to rewrite code?"*

- Apprehension

- *"I'm always paranoid that when someone prunes a language they'll take out something I always use. I hope those guys are careful. "*

- Trepidation
- *"Ow! Makes me feel like I'm a lobster and Larry Wall is dipping me in butter"*
- Irony
- *"A complete rewrite? This, from the man who advocates laziness in programmers?"*
- Mock paranoia
- *"O'Reilly is funneling massive kick backs to Larry Wall in exchange for making millions of O'Reilly Perl books obsolete. This way they get to "totally rewrite" the books and sell them to you again all over again. Think of the children! "*
- Real paranoia
- *"Doesn't Microsoft, the master of vaporware, somehow influence the Perl development group with a hefty financial arrangement?"*
- Confidence

- *"Change is scary. But it sounds like this change will be pretty transparent to programmers; perl will continue to get faster and better, still stay easy to use, and Larry Wall is still a little crazy. Well, it sounds like everything is right in the world. "*

- Cautious excitement

- *"I might just be biased on this from the heavy amount of web developement I do, but I'm anxiously awaiting the new version. I just hope it won't require 100% rewrite of my old scripts."*

- Disbelief

- *"Well, the Perl 6 thing was funny for a while, but I'm kinda unconvinced now. OK, Perl 6 is starting to look implausible. In fact, I'm convinced it's an elaborate practical joke..."*

- Mild hysteria

- *"...Oh, and if this isn't a joke - I quit."*

The Organization

- Meanwhile the movers and shakers were shoving and making
- Amailing list – `perl6-bootstrap` – was set up to coordinating planning of the development process
- A plan was drawn up
- A process was initiated
- The juggernaut started rolling

3. A Brief Present of Perl 6

- The people
- The purpose
- The plan
- The progress

The People

- The following roles were assigned
- Largely on the basis of willingness, past merit, and demonstrated ability
- Language designer: Larry Wall
- Project manager: Nathan Torkington
- Internals development: Dan Sugalski
- Quality Control: Michael Schwern
- Information repository: Adam Turoff
- Librarian: Ask Bjørn Hansen
- Corporate Relations: Dick Hardt
- Public Relations: `brian d foy`
- Perl 5 Maintenance: Jarkko Hietaniemi

Mission Statement

- Perl 6 will be a complete rewrite of Perl by the Perl community.
- Perl 6 will incorporate improvements to the language and to the internal API.
- The Perl 6 development process will be well managed, clearly understood by all involved, and open to any member of the Perl community who wishes to contribute.

The Plan

- A twelve step plan to a cleaner, healthier Perl

Step 1: Restructure the community

- Social self-engineering
- Initial mailing lists with chairs, charters, etc.
- Initial rules for growth and control of mailing lists

- Initial RFC structure
- Scheduled completion: On-going

Step 2: Brainstorming

- Identify issues, requirements, or wishes to use as topics for RFCs
- Seek input from Perl 5 developers, Perl users (sysadmins, web developers, application developers, CPAN authors...), vendors or distributors of Perl, other interested parties
- Draft RFCs
- Discuss and redraft RFCs
- Scheduled completion: 1 October 2000

Step 3: Design

- Larry evaluates the RFCs
- Puts them into his mental melting pot

- Produces a design specification for Perl 6
- Scheduled completion: 27 October 2000

Step 4: Infrastructure

- Engineer the implementation process
- Earlier stages mostly talk; this stage mostly code
- Initial mailing lists / working groups with goals and leaders
- Initial rules for growth and control of lists and tasks
- Quality assurance processes
- Change management processes
- Documentation processes
- Scheduled completion: 27 October 2000

Step 5: Analysis

- Flesh out design into hard requirements and specifications
- Subgroup may need to prototype implementations to facilitate this
- Scheduled completion: 31 December 2000

Step 6: Implementation

- Coding of the Perl 6 core
- Coding of the standard library
- Quality assurance and testing
- Change management
- Documentation
- Scheduled completion: Ongoing through 2001

Step 7: Alpha release

- Selection of testers
- Distribution
- Feedback mechanisms
- Scheduled: June 2001

Step 8: Beta release

- Wide distribution
- High volume feedback mechanisms
- Platform testing
- Not yet scheduled

Step 9: Liaison

- With vendors regarding distribution of Perl 6
- Not yet scheduled

Step 10: Final release

- Not yet scheduled

Step 11: Maintenance

- Of the Perl 6 core
- Of the standard library
- Of the documentation
- Ongoing quality assurance
- Scheduled completion: Ongoing

Step 12: Party On Dudes!

Progress to date

- Roles assigned (19 July 2000)
- Forums set up (25 July 2000)
- RFC submissions called for (29 July 2000)

- Requirements analysis complete (1 August 2000)
- RFCs finalized and archived(1 October 2000)
- Design proceeding
- So we're currently at Step 3

4. A Not-very-brief Future of Perl 6

- What we could see
- What we probably will see
- What we might see
- What we won't see

The Shape of Things To Come (Maybe)

- A whirlwind tour of the vast range of proposals put forward in the RFC process

- A brief whistle-stop at 133 of the 361 submissions through which Larry is now trekking

The Proposals

- Submitted in a two month period
- At least two revisions each
- That's close to 14 full proposals (re-)submitted every day for 9 weeks
- Many were hotly debated, and significantly modified as a result
- 29 were ultimate retracted
- Collectively they covered:
  - internals and implementation
  - general language syntax and semantics
  - data structures
  - flow-of-control constructs
  - errors and exceptions

- IO
- OO
- regexes
- subroutines
- Unicode issues
- licensing
- quality assurance
- changes to the standard library

- It's 133 submissions to the next topic, you've got a full stomach of coffee, half a pack of No-Doze, it's dark, and I'm not wearing sunglasses.

- Hold on to you hats...here we go...

Internals

- Mainly focussed on:
  - making the internals more maintainable
  - making it easier to connect other languages to Perl
  - reducing the core

- Unicode

Connecting with other languages

- RFC 32: A method of allowing foreign objects in perl
- RFC 334: Perl should allow specially attributed subs to be called as C functions
- RFC 121: linkable output mode
- RFC 270: Replace XS with the Inline module as the standard way to extend Perl.
- Seamless access between Perl and C for objects and subroutines
- Automated creation of appropriate glue (data format transformations, type mapping, etc.)
- The Inline proposal is based on the excellent CPAN module

Standardize support for filenames

- RFC 36: Structured Internal Representation of Filenames
- DWIM handling of cross-platform filenames
- Automatic directory separator translation

Arbitrary precision numbers

- RFC 43: Integrate BigInts (and BigRats) Support Tightly With The Basic Scalars
- Transparent promotion of built-in numbers
- int → floating point (as now)
- floating point → BigInt or BigRat (as appropriate)
- Invisible to programmer (just DWIMs)

Clean up the core

- RFC 125: Components in the Perl Core Should Have Well-Defined APIs and Behavior
- RFC 323: Perl's embedding API should be simple
- Specify clean and simple APIs for all internal data types
- OO model (but probably not OO implementation)
- Document properly

Slim the core

- RFC 146: Remove socket functions from core
- RFC 155: Remove mathematic and trigonomic functions from core binary
- RFC 230: Replace format built-in with format function
- Put them in modules

- May or not be autoloaded on demand
- The last of these RFCs is based on the Text::Reform CPAN module (formerly part of Text::Autoformat)
- Substantial changes to the way formats work:
  - subroutine based (i.e. run-time)
  - lexically scoped
  - more powerful
  - more configurable
  - re-entrant

Better error messages

- RFC 214: Emit warnings and errors based on unoptimized code
- Compiler sometimes optimizes away the actual source of a problem
- Or replaces it with something faster
- So error message refers to code that doesn't seem to exist in the source

- Make error messages relate to the source, not the executable

Unicode

- RFC 294: Internally, data is stored as UTF8
- RFC 295: Normalisation and unicode::exact
- RFC 312: Unicode Combinatorix
- Define the One True Internal Format that all string data is converted to
- Data is automatically normalized (i.e. "combining characters" are combined to equivalent single codes)
- Implies eq compares the *meaning* of a string encoding, not the raw bytes
- But manually overridable via pragma

Compilation

- RFC 301: Cache byte-compiled programs and modules
- RFC 310: Ordered bytecode
- RFC 270: Replace XS with the Inline module as the standard way to extend Perl.
- Cache the internal byte-codes (or object code) of a program or module around for next time
- Possibly in a format that allows JIT or parallel loading

Language proposals

- Tend to be more tightly focussed on a single construct
- A vast assortment of suggestions
- From fixing very small annoyances

- To major new syntax and styles of programming

Guiding principle

- RFC 28: Perl should stay Perl.
- This is a golden opportunity to change everything
- Let's *not* take it

Fix iterators

- RFC 136: Implementation of hash iterators
- Fix `each` so it DWIMs with this:
- ```
  while ($key1 = each %hash) {
   while ($key2 = each %hash) {
        print $hash{$key1}^$hash{$key2}
   }
  }
  ```

Time

- RFC 7: Higher resolution time values
  ```

- Why throw away information?
- Have a floating point return value from `time()`
- Former behaviour still available via `int(time())`

Minimizing global variables

- RFC 17: Organization and Rationalization of Perl State Variables
- Spring clean them
- Throw out the ones no-one uses any more (e.g. `$[`, `$*`, `$#`)
- Put the rest inside subroutines
- Or in nice safe lexical scopes

The `local` operator

- RFC 19: Rename the local operator
- Works better in Latin (where it's *"loco"*)

- It really means *"install another variable* in loco parentis *of the named global, until control leaves this scope"*

- So there's very little "local" about it.

- But what to call it?

- Suggestions:
  - `now`
  - `save`, `dynsave`, `saveval`, `saverestore`
  - `temp`, `savetemp`, `tempsave`, `scopetemp`, `tempval`
  - `current`
  - `scratchpad`
  - `deliver`
  - `preserve`
  - `pushval`
  - `contain`
  - `detach`
  - `revalue`

- `let`

- Larry suggested `temporarily`, as in:

- `temporarily $x = 7;`

- Anyone know a good five-letter synonym???

Overloading

- RFC 20: Overloadable && and ||

- Currently (almost) the only operators whose behaviour cannot be redefined

- But without them it's hard to implement new "algebraic" classes

Context detection

- RFC 21: Subroutines: Replace wantarray with a generic want function

- Extended awareness of context

- Many more contexts to be aware of:

- 'HASH'
- `%hash = func();`
- 'STRING'
- `$val = $hash{func()};`
- `print func();`
- 'NUMBER'
- `$val = func() * 2;`
- `$val = sin(func());`
- `$val = $array[func()];`
- 'INTEGER'
- `$val = $array[func()];`
- `$val = "str" x func();`
- 'BOOLEAN'
- `if ( func() ) {...}`
- `$val = func() || 0;`
- 'SCALARREF'
- `$val = ${func()};`
- 'ARRAYREF'

- `func()->[$index];`
- `push @{func()}, $val;`
- 'HASHREF'
- `func()->{key};`
- `@keys = keys %{func()};`
- 'OBJREF'
- `func()->method();`
- 'CODEREF'
- `func()->();`
- `&{func()}();`
- 'IOREF'
- `print {func()} @data;`
- 'LVALUE'
- `func() = 0;`
- 'RVALUE'
- `$val  = func();`
- `@vals = func();`

- Also, could retrieve how many list return values are expected, so only need to generate that many:

- ```
  ($line1, $line2, $line3) =
  get_data();
  ```

Switch statement

- RFC 22: Control flow: Builtin switch statement

- Swiss Army Switch

- Based on CPAN Switch.pm module

  - ```
    switch ($val) {
    ```
  - ```
      case 1          { print "number 1" }
    ```
  - ```
      case "a"        { print "string a" }
    ```
  - ```
      case [1..10,42] { print "number in
    list" and next }
    ```
  - ```
      case (@array)   { print "number in
    list" }
    ```
  - ```
      case /\w+/      { print "pattern" }
    ```
  - ```
      case qr/\w+/    { print "pattern" }
    ```

```
• case (%hash)    { print "entry in hash" }
• do_something_here();
• case (\%hash)   { print "entry in hash" }
• case (\&sub)    { print "arg to subroutine" }
• else            { print "previous case not true" }
• }
```

DWIMier comparisons

- RFC 25: Operators: Multiway comparisons
- `if (1 < $x < 10) {...}`
- Currently an error.
- Make it work just like the writer intended

Making list returns more managable

- RFC 37: Positional Return Lists Considered Harmful
- RFC 259: Builtins : Make use of hashref context for garrulous builtins
- `@context = caller(1);`
- Anyone know which element of `@context` tells you if you're inside an `eval`?
- `if ( (caller(1))[6] ) { print "in eval" }`
- Let `caller` and other such routines (e.g. `stat`, `localtime`) detect a 'HASHREF' context and return data that way instead:
- `if ( caller(1)->{ eval} ) { print "in eval" }`

Module termination

- RFC 55: Compilation: Remove requirement for final true value in require-d and do-ed files
- Almost no-one ever puts anything except `1` at the end of a module.
- So remove the need to
- Handle failure by exceptions instead
- Might produce the single biggest reduction in unneeded grief of any proposal.

Standardize package name separator

- RFC 71: Legacy Perl $pkg'var should die
- Only allow `$pkg::var` in Perl 6
- Doesn't look good for the D'uh module
- Or for programming in Klingon

- *Hu'tegh baQa' ghay'cha!*

Constants

- RFC 83: Make constants look like variables
- `my $pi : constant = 3;`

Configuration

- RFC 114: Perl resource configuration
- A `~/.perlrc` file
- Source code (most likely `use`'s) that is always executed before any script.
- For example:
- ```
  use strict;           # always paranoid
  use warnings 'all';   # always really paranoid
  use Coy;            # but sensitive
  ```
  ```

B&D

- RFC 140: One Should Not Get Away With Ignoring System Call Errors
- RFC 278: Additions to 'use strict' to fix syntactic ambiguities
- More discipline!
- `use strict 'system'`
- Kills you if you throw away return values of system calls
- `use strict 'words'`
- Kills you if you use *any* bareword (including class names and unparenthesized subroutines)
- `use strict 'objects'`
- Kills you unless you em-brace indirect objects
- `use strict 'syntax'`
- Kills you if you even breathe wrongly whilst you're coding

Loop extraction of multiple values

- RFC 173: Allow multiple loop variables in foreach statements

- ```
  foreach my ($x, $y, $z) (@list) {
    ...
  }
  ```

- Iterate variables two or more at a time

- If list was built lazily, would also solve hash iterator problem:

- ```
  foreach my ($key1, $val1) (%hash) {
   foreach my ($key2, $val2) (%hash) {
        print $hash{$key1}^$hash{$key2}
   }
  }
  ```

Remove vestigals

- RFC 195: Retire chop()

- ...because `chomp` is what you almost certainly want.
- And there always: `substr($str,-1,1,"");`

More DWIMity from common tests

- RFC 213: rindex and index should return true/false values
- RFC 221: system() should return useful values
- `index` and `rindex` currently return `0` if match is at first position
- Breaks a boundary case of a useful idiom:
- `if (index($str,$substr)) {...}`
- Return `"0 but true"` instead to fix that
- `system` returns the command return value
- Typically a Unixish 0-on-success
- Should return true so we can write:

- `system($cmd) || die;`
- instead of the just-plain-weird:
- `system($cmd) && die;`

String interpolation

- RFC 252: Interpolation of subroutines
- RFC 237: hashes should interpolate in double-quoted strings
- RFC 222: Interpolation of object method calls
- `print "Today is &date()\n";`
- `print "The data is %data\n";`
- `print "My name is $self->name()\n";`

Source filters

- RFC 264: Provide a standard module to simplify the creation of source filters

- Source pre-filters not yet widely used
- Current interface (Filter::Exec::Call CPAN module) very powerful and configurable
- But too complex for average module creator
- CPAN module Filter::Simple proposed as the solution

Better input control

- RFC 285: Lazy Input / Context-sensitive Input
- Use extended context information to determine how much data to suck in:
- `($x, $y, $z) = <SRC>`
- Would note finite list context and only read three lines

Alternative front-ends

- RFC 329: use syntax

- Compile-time selection of parser:
- `use syntax 'Perl5';`
- `use syntax 'Latin';`
- `use syntax 'Python';`

Vector processing

- RFC 82: Arrays: Apply operators element-wise in a list context
- RFC 90: Arrays: merge() and unmerge()
- RFC 91: Arrays: part and flatten
- RFC 116: Efficient numerics with perl
- RFC 117: Perl syntax support for ranges
- RFC 148: Arrays: Add reshape() for multi-dimensional array reshaping

- RFC 202: Arrays: Overview of multidimensional array RFCs
- RFC 203: Arrays: Notation for declaring and creating arrays
- RFC 204: Arrays: Use list reference for multidimensional array access
- RFC 205: Arrays: New operator ';' for creating array slices
- RFC 206: Arrays: @#arr for getting the dimensions of an array
- RFC 207: Arrays: Efficient Array Loops
- RFC 272: Arrays: transpose()
- Heavy multi-part proposal on data structures
- For heavy mathematical usages (i.e. PDL)
- Heavy-duty multidimensional arrays

- Heavy

Exceptions

- RFC 80: Exception objects and classes for builtins
- RFC 88: Omnibus Structured Exception/Error Handling Mechanism
- RFC 119: Object neutral error handling via exceptions
- Various approaches to providing a real OO exception mechanism
- Most suggest separate `try` and `catch` keywords to replace `eval {...}`

Foreach loop counting

- RFC 120: Implicit counter in for statements, possibly $#
- C-like `for`'s are ugly and unPerlish
- But occasionally necessary:

- ```
  for ($i = 0; $i <= $#array; $i++) {
   $array[$i]->getline;
   $array[$i]->parseline;
   $array[$i]->printline;
   $array[$i]->index = $i;
  }
  ```

- Provide an automagic lexically scoped punctuation variable that tracks the iteration number (from zero):

- ```
  foreach my $object  (@array) {
   $object->getline;
   $object->parseline;
   $object->printline;
   $object->index = $#;
  }
  ```

Slicker I/O

- RFC 311: Line Disciplines

- Real control over input and output processes

- Dictate how line endings are parsed

- Alter the buffering behaviour of the stream
- Interpose coding translations (to / from Unicode or EBCDIC)
- Interpose compression / decompression

Net support

- RFC 100: Embed full URI support into Perl
- ```
  my $handle = open
  'http://dev.perl.org/rfc/100.html'
   or die;
  ```

Threads

- RFC 178: Lightweight Threads
- RFC 185: Thread Programming Model
- Fix the threading model
- Favour micro-fork ("ithreads") model over shared-state model ("pthreads")
  ```

- Shared variables must be explicitly declared

Object Oriented Perl

- More declarative class specifications
- Cleaner and richer method dispatch semantics
- Built-in (optional) encapsulation
- Integrate OO and tie mechanisms
- Make good things easier and mistakes harder

Guiding Principle

- RFC 137: Overview: Perl OO should not be fundamentally changed.
- My view
- *"It ain't broke."*
- No need to "fix", only strengthen

Explicit class declaration syntax

- RFC 95: Object Classes
- A `class` keyword separate from `package`
- Declarative classes (à la Java, C++, Eiffel, Ada, etc.)

Alleviate the $_[0] annoyance

- RFC 152: Replace invocant in @_ with self() builtin
- RFC 223: Objects: use invocant pragma
- Two different takes on allowing the object reference to be autoextracted
- Both allow this:
- ```
  package MyClass;

  sub mymethod {
   if (self->{attr}) { ... }
  }
  ```
  ```

Pseudo-hashes must die!

- RFC 241: Pseudo-hashes must die!
- Pseudo-hashes must die!
- A failed experiment
- Foster the illusion of security
- Prone to nasty, subtle bugs

Simpler operator overloading

- RFC 159: True Polymorphic Objects
- Replace `use overload`
- Named methods: `STRING`, `PLUS`, `CONCAT`, etc.

Attribute access control

- RFC 163: Objects: Autoaccessors for object data structures

- Ability to declare read-only and write-only object attributes

Less holey blessing

- RFC 187: Objects : Mandatory and enhanced second argument to bless
- `bless` must be called with class name as second argument
- No more base-class time-bombs
- Also, second argument would be smarter about determining class name
- Implicitly tries: `ref $_[1] || $_[1]`

Encapsulation

- RFC 188: Objects : Private keys and methods
- RFC 336: use strict 'objects': a new pragma for using Java-like objects in Perl

- Private, protected, and public attributes for hash-based objects.
- Based on the Tie::SecureHash CPAN module

Proper object set-up and clean-up

- RFC 189: Objects : Hierarchical calls to initializers and destructors
- Initializers and clean-up routines with standard names
- `BUILD` and `DESTROY`
- Class's `BUILD` called automagically when an object `bless`'ed
- All ancestral class's `BUILD` methods also called!
- Likewise calls to `DESTROY` become automatically hierarchical

Delegation

- RFC 193: Objects : Core support for method delegation
- "Has-a" semantics sometimes more appropriate than "is-a" semantics
- Would cause a method call on an object to be automatically forwarded to a particular attribute of that object
- Based on Class::Delegation CPAN module

Interfaces

- RFC 265: Interface polymorphism considered lovely
- Java-like interfaces to go with Perl 6's optional static typing

Better reflection

- RFC 335: Class Methods Introspection: what methods does this object support?
- Rather than writing:
- ```
  @methods = grep {defined
  &{"MyClass::$_"}} keys %MyClass::;
  ```
- you could write:
- ```
  @methods = $obj->methods();
  ```

Regular Expressions

- Other data sources
- Easier access to matched data
- More control during a match

More flexible partial matching

- RFC 93: Regex: Support for incremental pattern matching
- RFC 316: Regex modifier for support of chunk processing and prefix matching
- Allow patterns to match incomplete data
- Such as that taken from an input stream
- Under first proposal, pattern can request more input from a sub during matching
- Under second proposal, pattern can signal "premature end-of-data" and then allow user to add provide more data manually
- Possibly: `\*STDIN =~ /$pattern/`

Identifying extracted data

- RFC 110: counting matches

- RFC 150: Extend regex syntax to provide for return of a hash of matched subpatterns
- Named capturing brackets, so you don't have to count them anymore.
- Matched substrings returned as variables (first proposal) or a hash (second proposal)

Safer precompiled regexes

- RFC 276: Localising Paren Counts in qr()s
- Within a `qr/.../`, the back-reference `\1` would mean "the substring matched by the first capturing parenthesis *in the qr//*"
- Can then safely interpolate into larger patterns with their own capturing parentheses

Optimized match variables

- RFC 158: Regular Expression Special Variables
- Make `$`, `$&`, and `$'` lexically scoped

- Also add an explicit modifier to request they be set
- Now their overheads are only incurred by the regexes in which they're actually used

Assertions

- RFC 348: Regex assertions in plain Perl code
- Retarget the mysterious `(?{...})` construct as a zero-width boolean assertion
- Like `(?=...)` and `(?>...)`
- For example, instead of:
- `/^25\d|2[1-4]\d|1?\d{1,2}$/`
- you could write:
- `/(^\d+$)(?{$1<256})/`

Subroutines

- Fewer built-ins

- New declaration syntaxes
- Lazy evaluation
- Wrappers

Subroutine autoloading

- RFC 8: The AUTOLOAD subroutine should be able to decline a request
- RFC 232: Replace AUTOLOAD by a more flexible mechanism
- RFC 190: Objects : NEXT pseudoclass for method redispatch
- Currently the AUTOLOAD belonging to the current package or most-derived class is invoked and must contend with every possibility
- Give an AUTOLOAD the chance to reject the invocation and let some other AUTOLOAD elsewhere in the inheritance tree handle the call.
- *"Sorry, we don't deal with that here: try down the road"*

Currying

- RFC 23: Higher order functions
- Proposes a freaky declarative subroutine syntax
- Instead of:
- `$tree->apply( sub{ $max += $_[0] } );`
- can write:
- `$tree->apply( $max += ^_ );`
- Delivers enormous power and the ability to do effective functional programming in Perl
- Resultant subroutines "curry" like so:
- ```
  $add = ^_ + ^_;
  foreach (@values) { $_ = $add->($_,1)
  }

  $incr = $add->(1);
  foreach (@values) { $_ = $incr->($_)
  }
  ```
  ```

Coroutines

- RFC 31: Subroutines: Co-routines
- Subroutines that remember where you exited them
- Then resume from that point, next time you invoke them
- Original call's argument list and lexicals are preserved
- Ideal for user-defined iterators
- For example:

- ```
  package Tree;

  sub next_inorder ($self) {
   yield $self->{left}->next_inorder
        if $self->{left};
   yield $self;
   yield $self->{right}->next_inorder
        if $self->{right};
   return;
  }

  while (my $node = $root-
  >next_inorder()) {
   print $node->{data};
  }
  ```

- Or for mapping hashes:

- ```
  %newhash = map {
        yield  process_key($_);
        return process_val($_);
  } %oldhash;
  ```

Subroutine parameter lists

- RFC 57: Subroutine prototypes and parameters

- RFC 128: Subroutines: Extend subroutine contexts to include named parameters and lazy arguments
- RFC 160: Function-call named parameters (with compiler optimizations)
- Much more powerful parameter specification mechanism, including:
  - Ability to prototype any built-in function
  - Named parameters
  - Variadic parameters
  - Ability to specify alternate argument types in a given position of the argument list (a la `map` and `grep`)
  - Ability to specify lazy evaluation
- ```
  sub my_grep_node([&//$]filter, \@list: lazy) {
    bless {
          filter => $filter,
          list    => $list,
    }, 'GrepNode';
  }
  ```

Lazy evaluation

- RFC 123: Builtin: lazy
  ```

- Lazy evaluation of lists
- Lazy evaluation of argument lists
- Probably not automatic
- Explicitly requested:
- ```
  sub process (@data : lazy) {...}

  process( 1..1000000000 );
  ```
- ```
  sub process (@data) {...}

  process( lazy(1..1000000000) );
  ```

Subroutine overloading

- RFC 97 : Prototype-based method overloading
- RFC 256: Objects : Native support for multimethods
- Multiple subroutines with the same name, but different parameter lists (à la C++)

- Second proposal also offers multiple dispatch based on Class::Multimethods CPAN module

Subroutine wrappers

- RFC 194: Standardise Function Pre- and Post-Handling
- RFC 271: Subroutines : Pre- and post- handlers for subroutines
- Wrappers for subroutines and built-in functions
- Allows extension of existing functionality:
- ```
  pre set_temp {
   $_[0]-=32;
   $_[0]/=1.8;
  }
  post get_temp {
   $_[-1]*=1.8;
   $_[-1]+=32;
  }
  ```
  ```

- ```
  pre CORE::open {
   $_[1] = "lynx -source $_[1] |"
        if $_[1] =~ m{^http://};
  }
  ```

- Allows memoization:

- ```
  my %sin_cache;

  pre CORE::sin {
   $_[-1] = $sin_cache{$_[0]}
        if exists $sin_cache{$_[0]}
  }

  post CORE::sin {
   $sin_cache{$_[0]} = $_[-1];
  }
  ```

- Second proposal also provides Design-by-Contract programming support at no extra charge

Odds and Ends

- Meta-linguistic

- Legal

Expand the standard library

- RFC 260: More modules
- ...in the standard distribution
- Survey Perl community to see what's missing

Licensing

- RFC 211: The Artistic License Must Be Changed
- RFC 346: Perl6's License Should be (GPL|Artistic-2.0)
- A new Artistic License for Perl
- Unambiguous
- Unrestrictive
- Widely acceptable

- Commercially viable
- Legally watertight
- Still simple

The Dark Side

- Proposals that are very unlikely to be accepted
- Often because Larry has already expressed his disinclination
- Or because they're too unPerlish
- Or because they create more problems than they solve

Inferential type system

- RFC 4: type inference
- An ML-like, inferred, dynamic type system

Global variables seriously deprecated

- RFC 6: Lexical variables made default
- RFC 64: New pragma 'scope' to change Perl's default scoping
- `use strict 'vars'` active by default
- Using an undeclared variable creates a new lexical variable within the scope
- Widely opposed, especially by:
  - RFC 16: Keep default Perl free of constraints such as warnings and strict
  - RFC 330: Global dynamic variables should remain the default
  - RFC 106: lexical variables made default without requiring strict 'vars'

Arrays and hashs assimilated

- RFC 9: Highlander Variable Types

- RFC 341: Unified container theory
- *"Thar ken be onla wun (type)!"*
- Merge scalars, arrays, and hashes
- Only scalars left
- `$array[$index]` becomes an abbreviation for `$array->[$index]`
- Strongly opposed, since `$array = 7` would then wipe out the entire array

New comment markers

- RFC 102: Inline Comments for Perl
- RFC 5: Multiline Comments for Perl.
- `#< Would let you write a Perl comment that spans multiple lines >#`
- `#<or># push #<comments between># @arg1, #<and># @arg2;`

No more prefixes

- RFC 133: Alternate Syntax for variable names
- Get rid of $, @, and % prefixes
- Use bare identifiers to access variables
- Determine type of variable by context

No special names

- RFC 243: No special UPPERCASE_NAME subroutines
- No BEGIN, FETCH, AUTOLOAD, etc.
- ```
  use tie
    FETCH => sub {...},
    STORE => sub {...},
    # etc.
  ;
  ```
  ```

Slices default to end-of-container

- RFC 282: Open-ended slices
- `@list[2..]` instead of `@list[2..$#list]`
- `@list[..2]` instead of `@list[0..2]`
- `@list[..]` instead of `@list[0..$#list]`

Pseudo-scalar access to hash entries

- RFC 342: Pascal-like "with"
- ```
  with (%hash) {
    $key1 = <>;         # $hash->{key1} = <>;
    $key2 = func();     # $hash->{key2} = func();
  }
  ```

Pass-by-value subroutines

- RFC 344: Elements of @_ should be read-only by default
- Pass-by-reference would have to be explicitly requested
  ```

OOO Perl

- RFC 352: Merge Perl and C#, but have default Main class for scripting.
- RFC 73: All Perl core functions should return objects
- RFC 161: Everything in Perl becomes an object.
- Object Oriented Only Perl.

New features for pack and unpack

- RFC 142: Enhanced Pack/Unpack
- RFC 246: pack/unpack uncontrovercial enhancements
- RFC 247: pack/unpack C-like enhancements
- RFC 248: enhanced groups in pack/unpack
- RFC 249: Use pack/unpack for marshalling

- RFC 250: hooks in pack/unpack
- Evidently they're not scary enough yet.

Less punctuated regexes

- RFC 164: Replace =~, !~, m//, s///, and tr// with match(), subst(), and trade()
- Functional style of matching
- Proposed syntax definitely cleaner (less line-noise)
- But almost certainly too radical a change
- *"...soulless..."*

Context-aware lvalue subroutines

- RFC 132: Subroutines should be able to return an lvalue
- RFC 149: Lvalue subroutines: implicit and explicit assignment

- RFC 154: Simple assignment lvalue subs should be on by default

- Let's break what little encapsulation Perl has by passing lvalue subroutines the lvalue they're being assigned to

Malleable syntax

- RFC 309: Allow keywords in sub prototypes

- ```
  sub locate ($, "in", @) {...}

  # and then

  my $xloc = locate $x in @list;
  ```

- Probably tough on the parser

Finality

- RFC 141: This Is The Last Major Revision

- Proposes Perl slowly converge on version 2π (6.283185307179586....)
  ```

Bestiality

- RFC 343: New Perl Mascot
- Abandon the Camel
- Because something like the Lithuanian tree vole isn't already copyrighted by Evil Multinational Publishers Who Secretly Control Perl For Their Own Financial Gain And Only License The Mascot's Image To Their Own Lackeys!!!!!

5. Making sense of the deluge

- If your head is now spinning, bear in mind that weve only just skimmed slightly more than 1/3 of the RFCs
- Larry has to assail, assay, assess, assimilate, and associate twice as many again
- In excruciating detail
- Trying to see implications and interrelationships as well

- That's why the design phase is already stretching far longer than the schedule allowed for
- There's not much to be done about that: the task is herculean and we've *already* got our best man on it
- We now wait for him to point the way

Through a Glass, Darkly

- But that needn't stop us from speculating
- Prognostications
- Predictions
- Outright guesses

Prognostications

- These common questions-and-answers about the shape of Perl 6 are taken from Larry's various speeches and interviews.
- The Perl 6 features they suggest should be considered "probable" (because of Rule #1)...

- ...but no more than "probable"
  (because of Rule #2)

Is Perl 6 going to be in C++?

- *"Maybe. Chip has a lot of experience with thinking about Perl and C++ and we intend to use the lessons he's learned one way or another."*

Will Perl 6 have specs?

- *"Yeah. I don't know how strict a spec it will be from the language design point of view. I'm not really big on that sort of spec and there is some value to using the reference implementation approach..."*

Will you rename the local operator?

- *"Yeah, I think we ought to do that. It confuses people when we call it* `local`*."*

Will there be changes to Perl's OO model?

- *"We're going to be standardizing the way that objects are set up. Not that we're going to be undoing all the flexibility that you have right now, but you'll have some standard ways that will be easier and more efficient. Perl 5 by design made it easy to use objects, but not easy to write object classes. We'd like to make it easy to write object classes."*

What about garbage collection?

- *"When you see all the new languages coming out and they all have a garbage collector and that helps them fit together into browsers and things like that better, you start thinking, maybe we ought to think about that."*

- *"It's becoming obvious that we need a real garbage collector, either one of our own or we need to be able to relate to one that is supplied."*

How about multiline comments?

- *"Multiline comments in Perl. I'm sorry, I was brainwashed by the Ada rationale. Ada has comments to the end of the line and I think that if your editor can't put a row of sharps down the left, then you need a better editor."*

Will Perl 6 have strong typing?

- *"You'll be able to get nice declarations like this:*

```
my num $pi :const = 3;
```

- *And similarly if you have homogenous arrays or hashes, you'd like to be able to declare the type of all the elements and have compact storage..."*

What about threads?

- *"My current leaning is that I really like the new ithreads model because I think it simplifies the way you can think about it...[it's] is more like a fork inside the process. [But] I do expect that both models [ithreads and pthreads] will be there, up to and including sharing everything.*

Can you hint at any other language changes that you're considering?

- *"There's really no reason why formats should be in the core anymore. They should be in a module."*
- *"There are things that could be done perhaps to clean up [the] ambiguities of the indirect object syntax."*
- *"Ther are various languages with a cleaner object interface to their IO..."*
- *"You could have [optional] strict-type checking if you wanted."*
- *We intend to get rid of quite a few of those strange global variables.*
- *It would be great to have higher-resolution time values. I think they should be floating point."*

What will happen to Perl 5?

- *"...one thing I want to make clear about this Perl 6 activity is that we're not abandoning Perl 5 any time soon. We don't know how long this Perl 6 will take. There are still some people using Perl 4 out there."*

What about Perl 7?

- *One of the proposals, one of the RFCs was that Perl 6 should be the last version of Perl. The idea is that if we make Perl 6 sufficiently flexible, then there's no need for Perl 7, because you can always bend the full language into whatever you want it to be.*

Predictions

- These suggestions of likely features have been made by various members of the Perl Illuminati (most notable Nathan Torkington, Chris Nandor, and Kirrily Robert)

- They should be considered merely "possible"

- Fewer special global variables (especially punctuation variables)
- All deprecated features removed
- No more typeglobs
- Filehandles become objects
- String interpolation of subroutine and method calls
- Extended subroutine prototyping and optional type enforcement for subroutine parameters
- `wantarray` becomes a more generic `want`
- More applications development support: web, XML, CORBA
- Internals more modular and easier to maintain
- Rewrite of the Artistic License
- Perl's parser and lexer written in Perl, using Perl regular expressions

- Perl compiler and interpreter systems will be able to emit C, Java, and C# code
- XS replaced with something much easier to use (possibly the Inline module)
- Completely object-oriented exception handling
- Overloadable logic operators
- `reduce` operator to facilitate functional programming styles.
- Full Unicode support for data and code, including the ability to declare lexically-scoped Unicode operators

Guesses

- These are my own considered views on what we might also see in Perl 6
- They should be considered "wild, utterly baseless speculation, no better than wish-fulfilment fantasies"

- Proper encapsulation of objects and support for Design-by-Contract
- Hierarchical constructors and destructors
- Method delegation and multimethod dispatch
- Higher-order functions
- Selective AUTOLOAD-ing of subroutines
- Overloadable && and || operators
- List-returning built-ins can also return named values in a hash
- Lazy evaluation of subroutine arguments
- Pattern matching on input streams
- Pseudo-hashes must die!

The final word

- I asked Larry what was the one message he most wanted me to convey to you

- He replied:

- *"My basic message right now is that we're going to take the time to do it right. What we want Perl to be in 2 years is a language that will be good for another 20 years."*

- That's something we can all look forward to

6. Resources

- `http://www.perl.org/perl6/`
  (Offical Perl 6 site)

- `http://infotrope.net/opensource/software/perl6/`
  (Unoffical Perl 6 site)

- `http://dev.perl.org/`
  (Perl 6 repository)

- `http://www.perl.com/pub/2000/10/23/soto2000.html`
  (Larry's TPC4 speech)

- http://technetcast.com/
  tnc_play_stream.html?stream_id=375
  (Larry's Atlanta Linux Showcase speech)

- http://slashdot.org/
  articles/00/07/20/210229.shtml
  (Geekdom's reaction to the Perl 6 announcement)

- http://www.perl.com/
  pub/2000/11/perl6rfc.html
  (A critique of the RFC process)

- http://www.perl.com/
  pub/2000/11/jarkko.html
  (A rebuttal of the above critique)

- http://history.perl.com/
  (The Perl timeline)

- http://www.etla.org/retroperl/

  http://mirrors.valueclick.com/
  perl/really-ancient-perls

  http://mirrors.valueclick.com/
  backup.pause
(Archives of former versions of Perl)