The Rakudo Update

Джонатан Вортингтон

Український воркшоп «Перл мова» 2009

Is it finished yet?

Is it finished yet?

No

Нет

Hi

Will it be finished tomorrow?

The Rakudo Update

Will it be finished tomorrow?

No

Нет

Hi

Will it be finished by Christmas?

Will it be finished by Christmas?

Yes!

Да!

Так!

Will it be finished by Christmas?

Yes!

Да!

Так!

(We just don't know which Christmas…)

One year ago...

The Rakudo Update

The Rakudo Update

The Rakudo Update

The Rakudo Update

The Rakudo Update

A year later...

Overview

- Much more rigorous testing, which has led to greater stability
- A lot of new features implemented
- A lot of things that we were sort of cheating on now done correctly
- More people contributing patches
- More people writing Perl 6 programs and running them on Rakudo

Testing

Testing Really Matters

- Experience has shown that…
  - Whenever a new feature is added, tests must be added at the same time
  - Otherwise, it will get broken by some future change
- Many complex feature interactions
- Changes can have unexpected side-effects => tests highlight these

Perl 6 Specification Tests

- A growing suite of tests that Rakudo must pass to be able to declare itself as a conforming Perl 6 implementation
- Test scripts are written in Perl 6, using a Test.pm also written in Perl 6
- Implementations also have their own "sanity tests", which test the basic features needed to be able to run Test.pm

Example Perl 6 Specification Test File

```
use v6;
use Test;
plan 27;

#L<S03/Changes to Perl 5 operators/"x (which concatenates repetitions">

is('a' x 3, 'aaa', 'string repeat operator works on single character');
is('ab' x 4, 'abababab', 'string repeat operator works on multiple character');
is(1 x 5, '11111', 'number repeat operator works on number and creates string');
is('' x 6, '', 'repeating an empty string creates an empty string');
is('a' x 0, '', 'repeating zero times produces an empty string');
is('a' x -1, '', 'repeating negative times produces an empty string');

#L<S03/Changes to Perl 5 operators/"and xx (which creates a list)">

my @foo = 'x' xx 10;
is(@foo[0], 'x', 'list repeat operator created correct array');
is(@foo[9], 'x', 'list repeat operator created correct array');
is(+@foo, 10, 'list repeat operator created array of the right size');

...
```

Example Perl 6 Specification Test File

```
use v6;
use Test;
plan 27;

#L<S03/Changes to Perl 5 operators/"x (which concatenates repetitions">

is('a' x 3, 'aaa', 'string repeat operator works on single character');
is('ab' x 4, 'abababab', 'string repeat operator works on multiple character');
is(1 x 5, '11111', 'number repeat operator works on number and creates string');
is('' x 6, '', 'repeating an empty string creates an empty string');
is('a' x 0, '', 'repeating zero times produces an empty string');
is('a' x -1, '', 'repeating negative times produces an empty string');

#L<S03/Changes to Perl 5 operators/"and xx (which creates a list)">

my @foo = 'x' xx 10;
is(@foo[0], 'x', 'list repeat operator created correct array');
is(@foo[9], 'x', 'list repeat operator created correct array');
is(+@foo, 10, 'list repeat operator created array of the right size');

...
```

Example Perl 6 Specification Test File

```
use v6;
use Test;
plan 27;

#L<S03/Changes to PerlÂ 5 operators/"x (which concatenates repetitions">

is('a' x 3, 'aaa', 'string repeat operator works on single character');
is('ab' x 4, 'abababab', 'string repeat operator works on multiple character');
is(1 x 5, '11111', 'number repeat operator works on number and creates string');
is('' x 6, '', 'repeating an empty string creates an empty string');
is('a' x 0, '', 'repeating zero times produces an empty string');
is('a' x -1, '', 'repeating negative times produces an empty string');

#L<S03/Changes to PerlÂ 5 operators/"and xx (which creates a list)">

my @foo = 'x' xx 10;
is(@foo[0], 'x', 'list repeat operator created correct array');
is(@foo[9], 'x', 'list repeat operator created correct array');
is(+@foo, 10, 'list repeat operator created array of the right size');

...
```

Example Perl 6 Specification Test File

```
use v6;
use Test;
plan 27;

#L<S03/Changes to Perl 5 operators/"x (which concatenates repetitions">

is('a' x 3, 'aaa', 'string repeat operator works on single character');
is('ab' x 4, 'abababab', 'string repeat operator works on multiple character');
is(1 x 5, '11111', 'number repeat operator works on number and creates string');
is('' x 6, '', 'repeating an empty string creates an empty string');
is('a' x 0, '', 'repeating zero times produces an empty string');
is('a' x -1, '', 'repeating negative times produces an empty string');

#L<S03/Changes to Perl 5 operators/"and xx (which creates a list)">

my @foo = 'x' xx 10;
is(@foo[0], 'x', 'list repeat operator created correct array');
is(@foo[9], 'x', 'list repeat operator created correct array');
is(+@foo, 10, 'list repeat operator created array of the right size');

...
```

Fudge

- Test files are written in Perl 6
- During development of Rakudo, we can sometimes only run parts of a test file
  - Sometimes we just get the wrong answer => can mark the test "todo"
  - Other times, it causes a parse error or compiler failure => we never get to run any tests

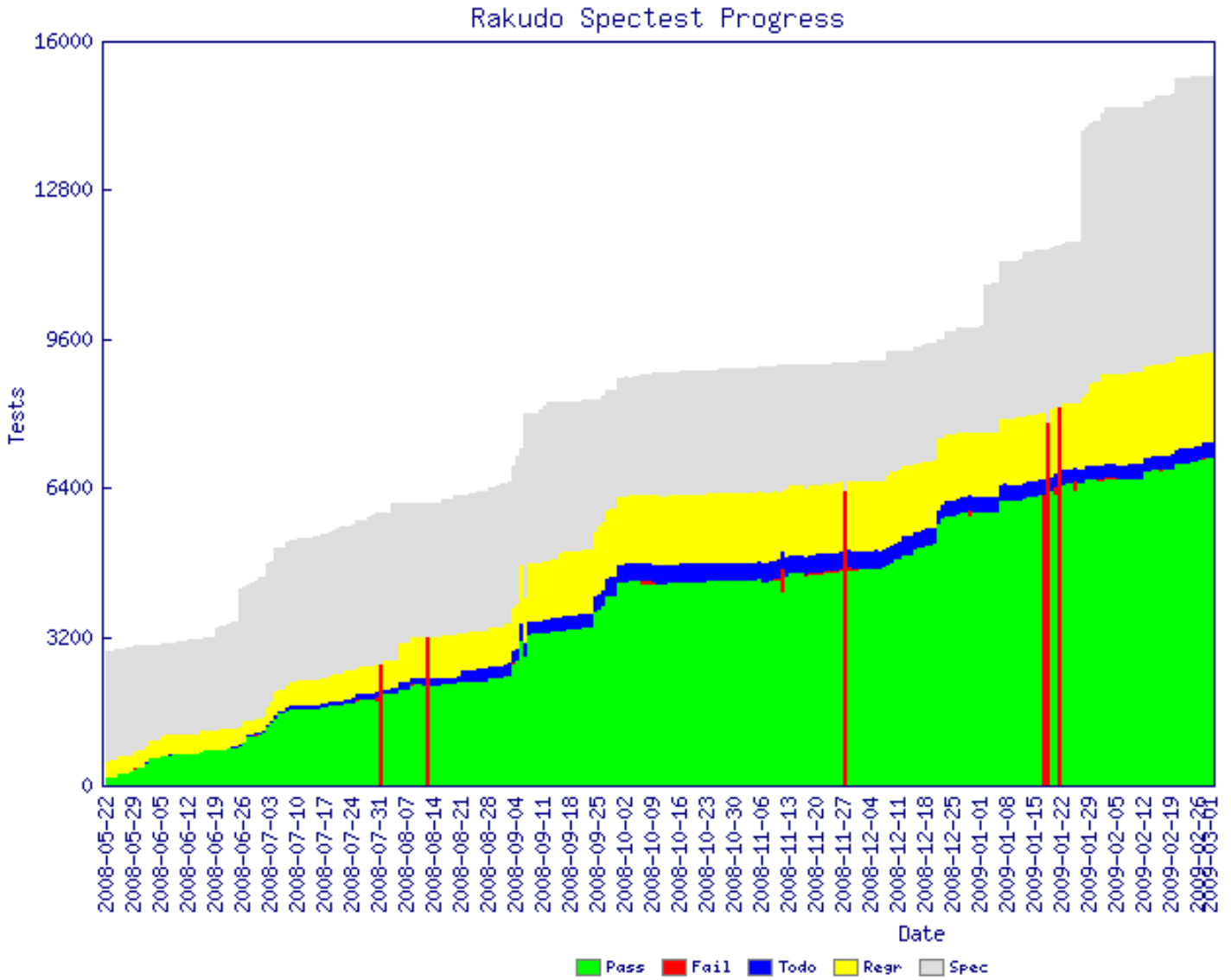
Fudge

- Insert "fudge" directives into a test file

```
#?rakudo skip 'unimpl undef++'
my $a = undef;
is($a++, 0, 'undef++ == 0');
```

- A pre-processor runs before the test suite, comments out the tests and inserts calls to "skip"
- Fudge works per implementation, so they can each fudge the tests as they need to

Tests And Passing Tests Over Time



7121 passing spec tests

Features

Oh, so many…

- Not time to discuss in detail even a fraction of the Perl 6 features that have been added in the last year
- Will pick a few of the most major features, as well as areas that have seen major advances
- One of them – multiple dispatch – will be discussed in my other talk

Array and Hash Slicing

- Rakudo now supports array and hash composers:

```
my @a = 1,2,3,4,5;
```

```
my %nums = one => 1, two => 2;
```

- It also lets you get and assign to slices of lists and hashes:

```
@a[2,3] = @a[3,2];
say @a.perl;     # [1, 2, 4, 3, 5]
```

```
%nums<one two> = 10, 20;
say %nums.perl; # {"one" => 10, "two" => 20}
```

Junction Auto-threading

- Junctions now auto-thread properly through function and method calls
- In the following example, the sub double is called three times, and a new junction of the results is created

```
sub double($x) {
    return 2 * $x;
}
my $j = 1 | 2 | 3;
say double($j).perl; # any(2, 4, 6)
```

Some meta-operators

- The reduction meta-operator works:

```
my @costs = 49.99, 10.50, 5.23;
say [+] @costs;  # 65.72
say [*] 1..10;   # 3628800 (10 factorial)
```

- Various of the infix hyper-operators and cross-operators are also implemented:

```
my @a = 1,2;
my @b = 3,4;
say (@a >>+<< @b).perl; # [4, 6]
my @c = 'a', 'b';
say (@a X~ @c).perl; # ["1a", "1b",
                     #  "2a", "2b"]
```

Much Progress In OO

- Overall, OO support is now much more stable, more conformant with the specification and far more complete
  - Read-only accessors really are
  - Array and hash attributes work
  - Initialization of attributes in the class
- Generally, many things that you'd expect to Just Work now mostly do

Role Advances

- Run-time mixing in of roles into existing objects now works

```
role Units {
    has Str $.unit is rw;
}
my $x = 42 but Units("cm");
say $x;       # 42
say $x.unit; # cm
```

- Parametric roles also implemented => the Perl 6 way to do generics

```
role Array[::TElements] { … }
```

Pointy Block Improvements

- Pointy blocks now work as r-values, giving you a nice way to write lambdas

```
my $ten_times = -> $s { say $s for 1..10 };
$ten_times("OH HAI"); # 10 lines of output
```

- Can also use the syntax for many other types of block now too

```
if try_to_get_input() -> $read {
    say "We read $read";
}
```

Many More Built-ins

- Many more built-in types, methods and functions are now available
- More operators are implemented and working
- Some of the IO classes are now in place, giving us file IO

Architecture

Largely The Same

- The Parrot Compiler Toolkit has had various extensions
- However, the overall architecture of the compiler is about the same as it was a year ago
- The existing set of AST nodes has continued to be sufficient
- Toolkit being used for other compilers

The Perl 6 "Setting"

- The "Setting" is what other languages call a prelude: the set of built-in types and functions
- Until recently, all in PIR
- We are now doing a two-stage build of the compiler:
  - Build the core
  - Use that to compile the Setting

Forthcoming Major Parsing Changes

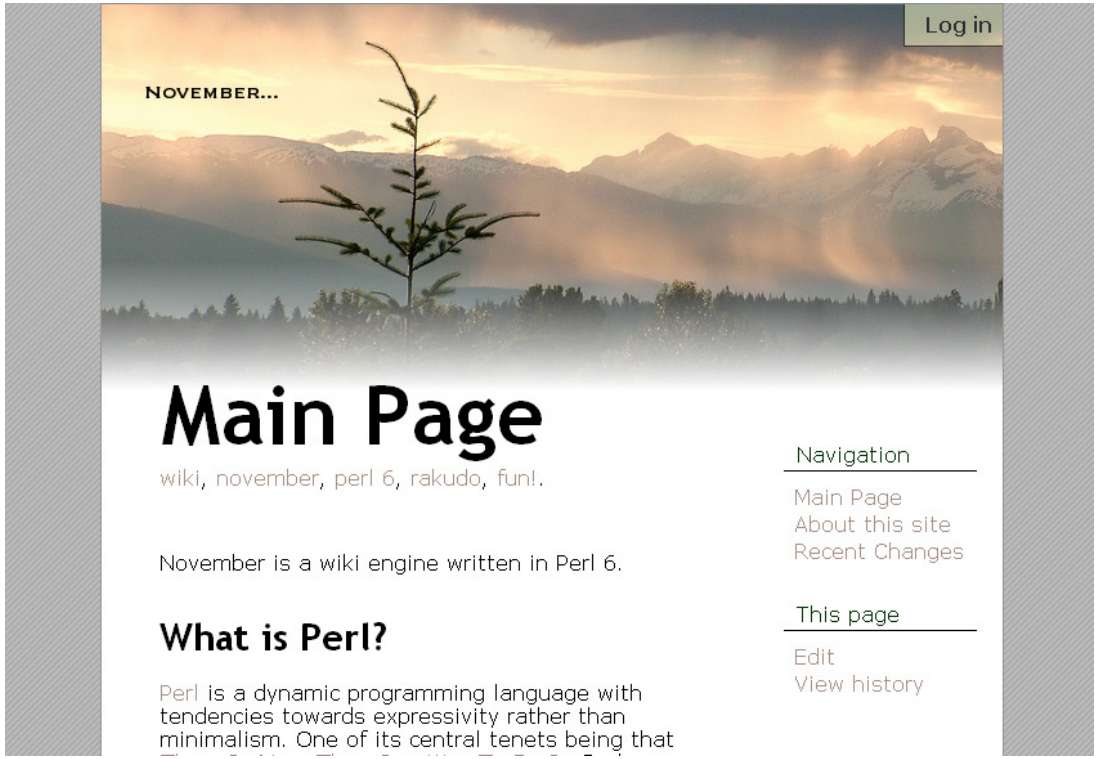
- While the AST and the toolkit beyond that stage will stay the same, the parsing engine will change substantially in the coming months
- Changes needed to fully support the official Perl 6 grammar
- Also expected to give a large performance increase – parsing today is one of our major bottlenecks

Projects Using Rakudo

The Rakudo Update

November

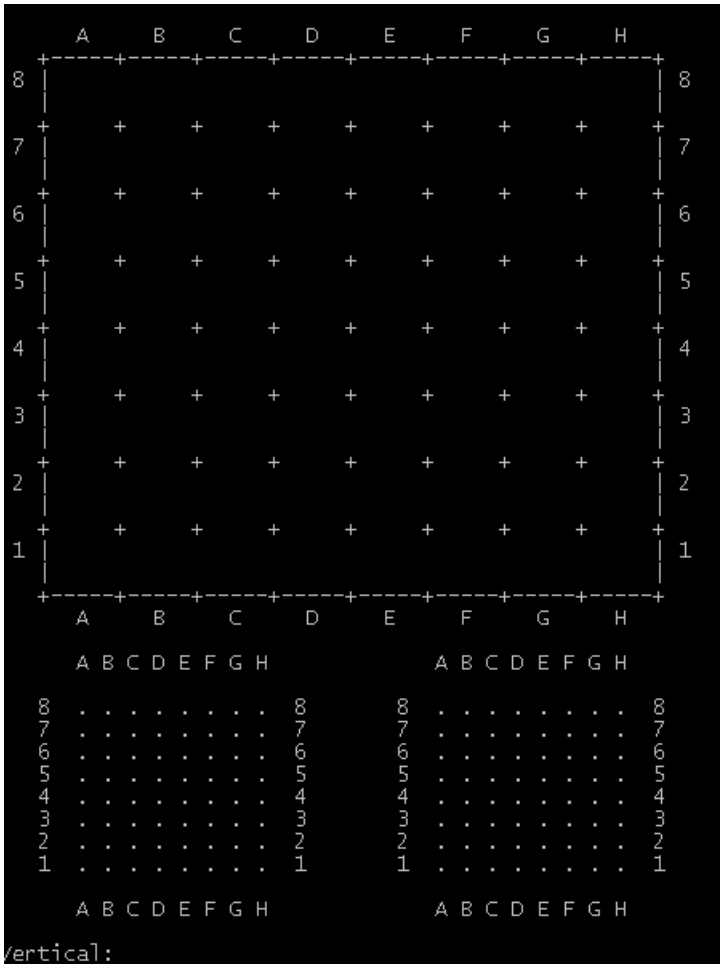
- Wiki written in Perl 6
- November developers have discovered and reported many bugs and helped motivate the Rakudo team to add more features.



The Rakudo Update

Druid

- A connection-oriented board game



SVG.pm

- An SVG module for Perl 6 (the implementation is beautiful)

```
use v6;
use SVG;

my $svg =
    :width(200), :height(200),
    circle => [
        :cx(100), :cy(100), :r(50)
    ]
;

say SVG.serialize($svg);
```

Getting Rakudo

Getting Rakudo

- Now hosted in a GIT repository git://github.com/rakudo/rakudo.git
- Build it (builds Parrot for you):

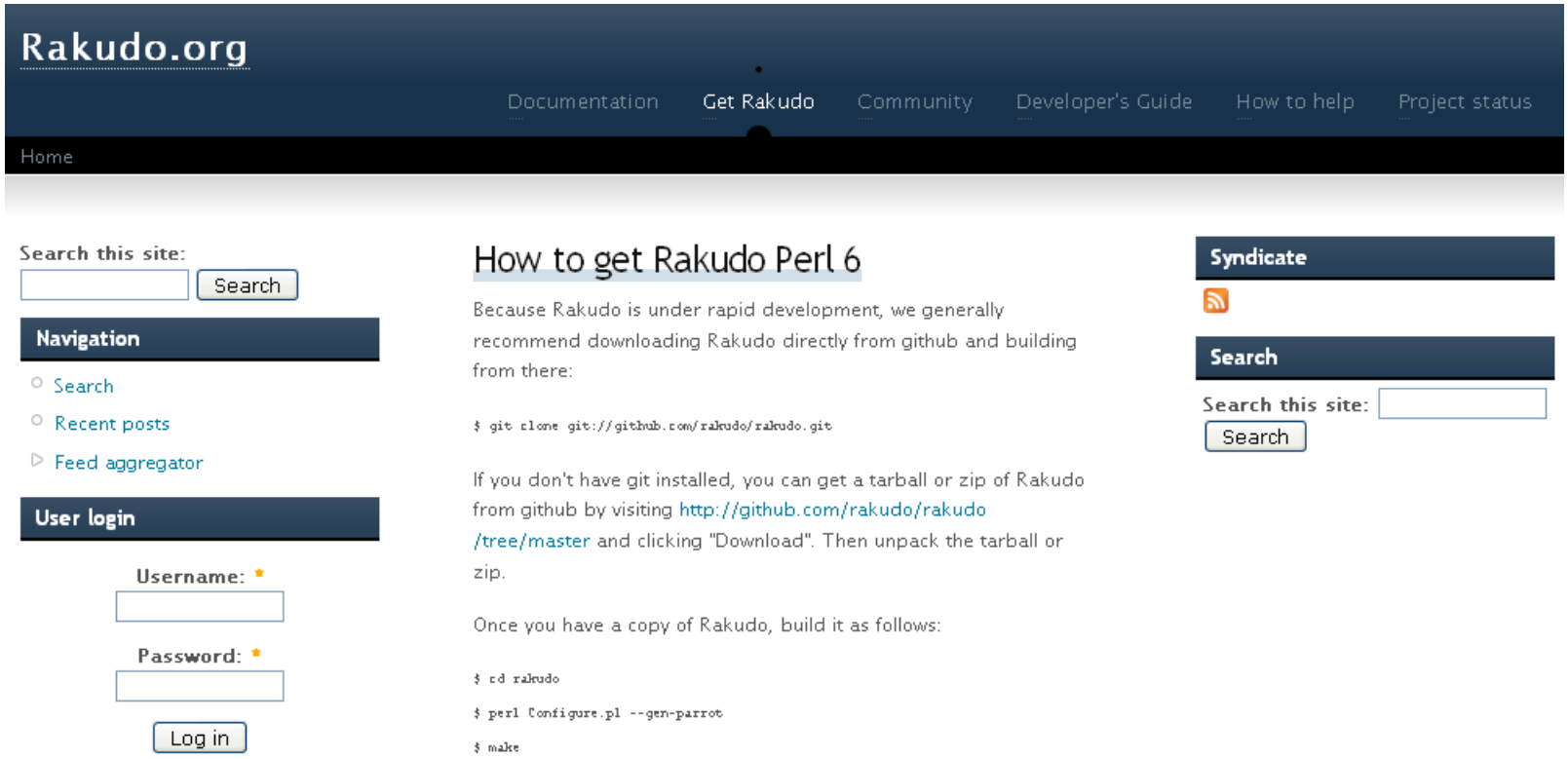
```
perl Configure.pl --gen-parrot
make perl6
```

- Run it on the command line, with a script or in interactive mode

```
perl6 -e "say 'Hello, world!'"
perl6 script.p6
perl6
```

The Rakudo Website

- www.rakudo.org
- Recently improved; now not just a blog



So really, how close are we?

It won't be 2009…

- Rakudo has made enormous progress in the last year, but there's still a lot more left to do
- Don't expect to have a production-ready release in 2009 (2010 is much more realistic) ☹
- Do expect to have Alpha/Beta style releases this year ☺

But you can use Rakudo today!

- The day I wrote these slides, I read a post on use.perl.org by someone who had just created their first useful Perl 6 program

"I know this is mind-bogglingly simple, but it's now a practical part of my development environment for work."

Get It, Try It, Break It

- Most bugs are discovered by people trying to build stuff with Rakudo
  - The November wiki project has been especially helpful in this regard
- Feedback about what people want and need from Rakudo helps motivate the compiler team too
- Looking forward to your bug reports ☺

Дякую!

Questions?