Morgan Stanley

Perl and complaints about $_

Balazs Fulop
Virtualization Engineering

What is Perl?

- a programming language
  - high-level
  - general purpose
  - interpreted
  - dynamic
- “the Swiss Army chainsaw of programming languages”
- “duct tape that holds the Internet together”

Complaint #1

- “I’m not doing everything I could be doing.”
- Are you sure you want to do everything you could be doing?

Complaint #1
“I’m not doing everything I could be doing.”

Portability across platforms

- C source code is portable – or is it?
- ```
  #ifdef _WIN32
  # include <windows.h>
  #else
  # include <unistd.h>
  #endif
  ```
  ```

Complaint #1
“I’m not doing everything I could be doing.”

Data structures

- e.g. get unique elements from a list of strings
- `my @uniq = do { my %seen; grep { not $seen{$_}++ } @list };`
- In other words…

```
my ( %seen, @uniq );
for my $elem (@list) {
  if ( not $seen{$elem} ++ ) {
    push @uniq, $elem;
  }
}
```

Complaint #1
“I’m not doing everything I could be doing.”

Memory management

- memory leak (lost pointer)
- invalid deallocation (double delete)
- garbage collection
- debugging mostly at runtime

Complaint #1
“I’m not doing everything I could be doing.”

Code safety

- invalid read / invalid write (segmentation fault)

```
i = a[-1];
$i = $a[-1];
```

- buffer overflow attacks

```
struct entitlements {
  int allow_read;
  int allow_write;
} user1;
my $user1 = {
  allow_read => undef,
  allow_write => undef,
};
```

Complaint #2

- “Interpreted languages are slow.”
- Do you know where the bottlenecks are in your system?

Complaint #2
“Interpreted languages are slow.”

http://rosettacode.org/

```
void bubble_sort (int *a, int n) {
    int i, s, t;
    while (1) {
        s = 0;
        for (i = 1; i < n; i++) {
            if (a[i] < a[i - 1]) {
                t = a[i];
                a[i] = a[i - 1];
                a[i - 1] = t;
                s = 1;
            }
        }
        if (!s) break;
    }
}
```

```
sub bubble_sort {
    for my $i (0 .. $#_) {
        for my $j ($i + 1 .. $#_) {
            @_[$i, $j] = @_[$j, $i]
              if $_[$j] < $_[$i];
        }
    }
}
```

3 ms vs 40 ms

Complaint #3

- “Why another language?”
- Why not? ;-)

Complaint #3
“Why another language?”

- New language – new way to look at things
- ```
  sub square_root {
    my %cache;
    return sub {
      unless ( defined $cache{$_[0]} ) {
        say "sqrt($_[0])";
        $cache{$_[0]} = sqrt $_[0];
      }
      return $cache{$_[0]};
    };
  }

  my $fn_sqrt = square_root;
  say $fn_sqrt->(4);
  say $fn_sqrt->(4);
  ```
  ```

Complaint #4

- “Weak typing is unsafe.”
- Did you know that C features weak typing?

Complaint #4
“Weak typing is unsafe.”

weak vs strong typing

- Strong typing

```
Integer a = 2;
String b = "2";
System.out.println(a.toString() + b); // 22
System.out.println(a + new Integer(b)); // 4
```

- Weak typing

```
my $a = 2;
my $b = "2";
say $a . $b; # 22
say $a + $b; # 4
```

- DWIM

Complaint #5

- “Dynamic typing is unsafe.”
- Does static typing eliminate the need for unit tests?

Complaint #5
“Dynamic typing is unsafe.”

dynamic vs static typing

- Static typing

```
class PekinDuck implements Duck {
  public void quack() { … }
}
// …
Duck duck = new PekinDuck();
duck.quack();
```

- Dynamic typing

```
my $duck = PekinDuck->hatch();
$duck->quack;
# eval { $duck->quack };
# $duck->quack if $duck->can('quack');
```

Complaint #5
“Dynamic typing is unsafe.”

duck typing

- “When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.”
- ```
  say "This bird is a duck." if
    $bird->can('walk') and $bird->can('swim') and $bird->can('quack');
  ```
- Less re-factoring of code, more focus on documentation
- Unit tests: required anyway
  ```

Complaint #6

- “Perl’s magic variables freak me out: `$!`”
- Have you considered to use the core module `English`?

Complaint #6
“Perl’s magic variables freak me out.”

- ```
  use English;
  my $filename = 'non-existent';
  open( my $fh, '<', $filename )
    or die "$filename: $OS_ERROR\n";
  ```
- ```
  ./test.pl
  non-existent: No such file or directory
  ```
- TMTOWTDI

Complaint #7

- “You can’t build real systems with Perl.”
- Do you know how many different technologies are typically used in the same project?

Complaint #7
“You can’t build real systems with Perl.”

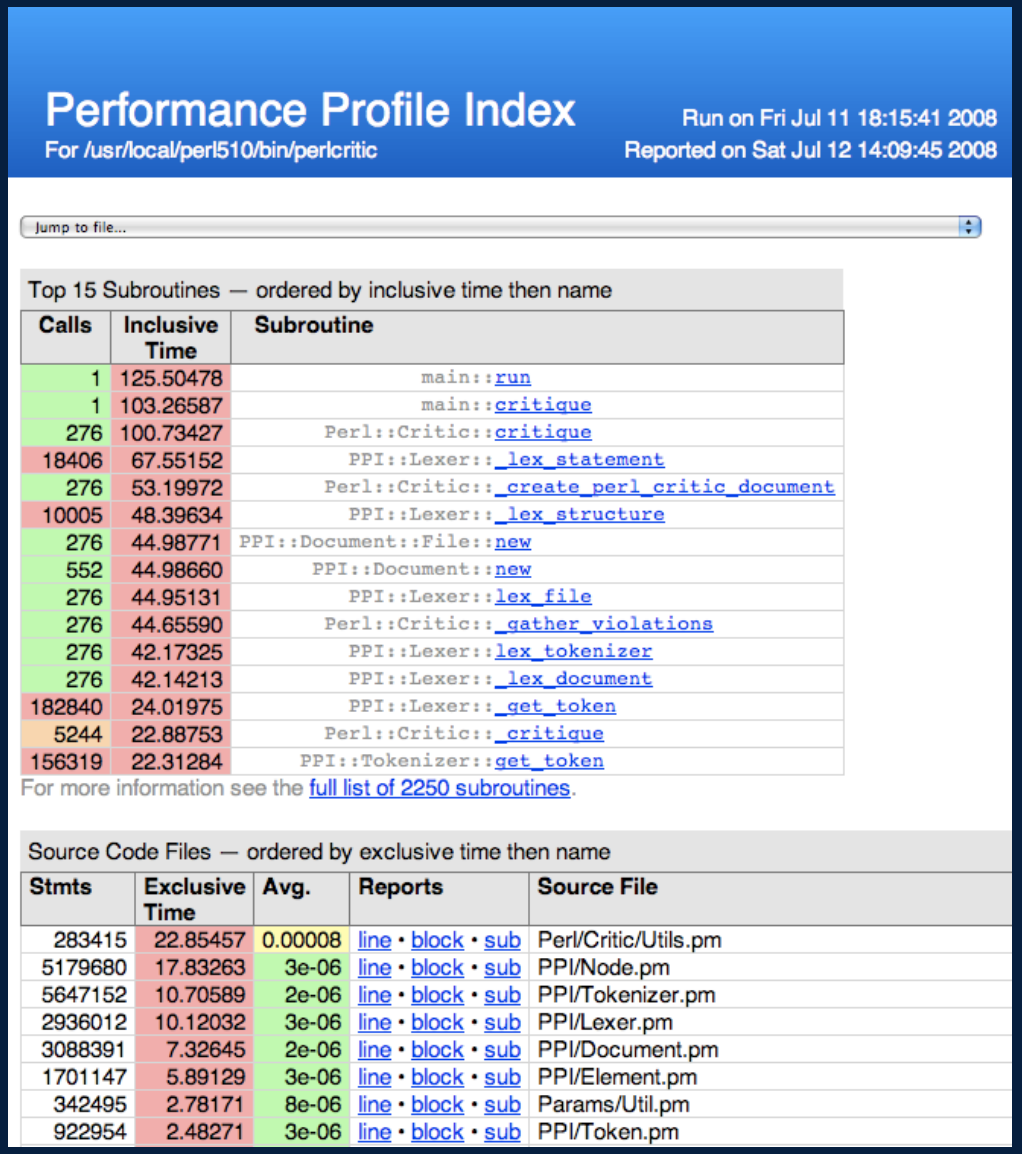
- Back-end in Java
- Functional tests, command-line management tools in Perl
- Front-end in C#

Complaint #8

- “Perl lacks the tools for profiling.”
- Have you tried `Devel::NYTProf`?

Complaint #8
“Perl lacks the tools for profiling.”

- number of calls made from or to any given subroutine
- time spent within a sub
- time spent making a call to an external sub
- “Hot Spots”



Performance Profile Index

For /usr/local/perl510/bin/perlcritic

Run on Fri Jul 11 18:15:41 2008
Reported on Sat Jul 12 14:09:45 2008

Jump to file...

Top 15 Subroutines — ordered by inclusive time then name

Calls	Inclusive Time	Subroutine
1	125.50478	main::run
1	103.26587	main::critique
276	100.73427	Perl::Critic::critique
18406	67.55152	PPI::Lexer::_lex_statement
276	53.19972	Perl::Critic::_create_perl_critic_document
10005	48.39634	PPI::Lexer::_lex_structure
276	44.98771	PPI::Document::File::new
552	44.98660	PPI::Document::new
276	44.95131	PPI::Lexer::lex_file
276	44.65590	Perl::Critic::_gather_violations
276	42.17325	PPI::Lexer::lex_tokenizer
276	42.14213	PPI::Lexer::_lex_document
182840	24.01975	PPI::Lexer::_get_token
5244	22.88753	Perl::Critic::_critique
156319	22.31284	PPI::Tokenizer::get_token

For more information see the full list of 2250 subroutines.

Source Code Files — ordered by exclusive time then name

Stmts	Exclusive Time	Avg.	Reports	Source File
283415	22.85457	0.00008	line • block • sub	Perl/Critic/Utils.pm
5179680	17.83263	3e-06	line • block • sub	PPI/Node.pm
5647152	10.70589	2e-06	line • block • sub	PPI/Tokenizer.pm
2936012	10.12032	3e-06	line • block • sub	PPI/Lexer.pm
3088391	7.32645	2e-06	line • block • sub	PPI/Document.pm
1701147	5.89129	3e-06	line • block • sub	PPI/Element.pm
342495	2.78171	8e-06	line • block • sub	Params/Util.pm
922954	2.48271	3e-06	line • block • sub	PPI/Token.pm

Complaint #9

- “Perl does not feature failsafe I/O.”
- Have you turned on `autodie`?

Complaint #9
“Perl does not feature failsafe I/O.”

- ```
  use autodie;
  my $filename = 'non-existent';
  open( my $fh, '<', $filename );
  ```
- ```
  ./test.pl
  Can't open 'non-existent' for reading: 'No such file or directory' at
  ./test.pl line 9
  ```
- Note: `die` is the same as `throw`, `eval BLOCK` is the same as `try`

Complaint #10

- “Perl code is unreadable.”
- How about commenting your code?

Complaint #10
“Perl code is unreadable.”

- ```
  say join ' ', /(..?)/g
    for ( unpack 'H*', $msg ) =~ /(.{1,32})/g;
  ```
- In other words…

```
# get ascii hex representation of the bytestream in $msg
my $msg_in_hex = unpack 'H*', $msg;

# cut $msg_in_hex into max 32 characters long chunks
# each chunk will represent at most 16 bytes from the original stream
my @chunks = ( $msg_in_hex =~ /(.{1,32})/g );

# for each chunk, print the 2-char-long hex representations of
# the bytes, separated by spaces, ending in a newline
for my $chunk ( @chunks ) {
  my @chunk_bytes = ( $chunk =~ /(..?)/g );
  say join ' ', @chunk_bytes;
}
```
```

Complaint #10
“Perl code is unreadable.”

- Or you could say…

```
# get ascii hex representation of the bytestream in $msg
# cut $msg_in_hex into max 32 characters long chunks
# each chunk will represent at most 16 bytes from the original stream
# for each chunk, print the 2-char-long hex representations of
# the bytes, separated by spaces, ending in a newline

say join ' ', /(..?)/g
  for ( unpack 'H*', $msg ) =~ /(.{1,32})/g;
```