WELCOME TO CPERL

perl11: 5 + 6 = 11

http://perl111.org/cperl/
QUO VADIS?

perl5+6 dev workshop 2011
Orlando 2013
PERL IS NOT DEAD, IT IS A DEAD END

Stevan Little
Orlando Perl Workshop 2013
stevan.little@iinteractive.com
Mouse not Moose
CPERL

• with classes (types)
• compiler support
• continuation of perl development
• company-friendly
CPERL

- Features
- Bugfixes, Security
- In Work
- Policies
FEATURES

• 44 new major features in the last 2 years

• cperl-5.22.1 (Feb. 2015) - 5.27.1

• the only full-time perl5 core developer

• ~10x more than p5p
FEATURES

• ~20% faster (with signatures and modern perl 2x faster)

• dramatically less memory (many helpers rewritten in XS)

• protection from p5p policies and destruction
  removal of lexical $_, use encoding, signatures, for (qw()) {}, …

• professional and public development, no mailinglist, no major features directly to master

• cperl: 90% of perl5 commits merged, perl5: 0% of cperl commits merged
### PerlBench results from reini at 2017-08-08 09:29:37

<table>
<thead>
<tr>
<th>Function</th>
<th>5.14</th>
<th>5.22</th>
<th>5.22c</th>
<th>5.24</th>
<th>5.24c</th>
<th>5.26</th>
<th>5.26c</th>
<th>5.27c</th>
</tr>
</thead>
<tbody>
<tr>
<td>arith/mixed</td>
<td>100</td>
<td>105</td>
<td>113</td>
<td>119</td>
<td>129</td>
<td>114</td>
<td>118</td>
<td>120</td>
</tr>
<tr>
<td>arith/trig</td>
<td>100</td>
<td>90</td>
<td>118</td>
<td>118</td>
<td>109</td>
<td>115</td>
<td>111</td>
<td>109</td>
</tr>
<tr>
<td>array/copy</td>
<td>100</td>
<td>143</td>
<td>130</td>
<td>136</td>
<td>142</td>
<td>190</td>
<td>189</td>
<td>196</td>
</tr>
<tr>
<td>array/foreach</td>
<td>100</td>
<td>140</td>
<td>135</td>
<td>197</td>
<td>187</td>
<td>187</td>
<td>219</td>
<td>183</td>
</tr>
<tr>
<td>array/index</td>
<td>100</td>
<td>129</td>
<td>141</td>
<td>153</td>
<td>144</td>
<td>159</td>
<td>159</td>
<td>155</td>
</tr>
<tr>
<td>array/pop</td>
<td>100</td>
<td>138</td>
<td>159</td>
<td>183</td>
<td>179</td>
<td>198</td>
<td>205</td>
<td>214</td>
</tr>
<tr>
<td>array/shift</td>
<td>100</td>
<td>127</td>
<td>143</td>
<td>146</td>
<td>161</td>
<td>171</td>
<td>172</td>
<td>170</td>
</tr>
<tr>
<td>array/sort-num</td>
<td>100</td>
<td>159</td>
<td>146</td>
<td>156</td>
<td>160</td>
<td>170</td>
<td>155</td>
<td>156</td>
</tr>
<tr>
<td>array/sort</td>
<td>100</td>
<td>134</td>
<td>133</td>
<td>131</td>
<td>134</td>
<td>137</td>
<td>135</td>
<td>138</td>
</tr>
<tr>
<td>call/0arg</td>
<td>100</td>
<td>143</td>
<td>107</td>
<td>169</td>
<td>163</td>
<td>191</td>
<td>144</td>
<td>195</td>
</tr>
<tr>
<td>call/1arg</td>
<td>100</td>
<td>124</td>
<td>101</td>
<td>170</td>
<td>156</td>
<td>193</td>
<td>154</td>
<td>166</td>
</tr>
<tr>
<td>call/2arg</td>
<td>100</td>
<td>155</td>
<td>169</td>
<td>186</td>
<td>191</td>
<td>206</td>
<td>253</td>
<td>256</td>
</tr>
<tr>
<td>call/9arg</td>
<td>100</td>
<td>143</td>
<td>142</td>
<td>167</td>
<td>153</td>
<td>159</td>
<td>189</td>
<td>188</td>
</tr>
<tr>
<td>call/empty</td>
<td>100</td>
<td>122</td>
<td>93</td>
<td>202</td>
<td>189</td>
<td>190</td>
<td>185</td>
<td>176</td>
</tr>
<tr>
<td>call/fib</td>
<td>100</td>
<td>113</td>
<td>114</td>
<td>126</td>
<td>136</td>
<td>146</td>
<td>120</td>
<td>139</td>
</tr>
<tr>
<td>call/method</td>
<td>100</td>
<td>112</td>
<td>109</td>
<td>134</td>
<td>136</td>
<td>138</td>
<td>141</td>
<td>123</td>
</tr>
<tr>
<td>call/wantarray</td>
<td>100</td>
<td>123</td>
<td>116</td>
<td>127</td>
<td>128</td>
<td>158</td>
<td>162</td>
<td>151</td>
</tr>
<tr>
<td>hash/copy</td>
<td>100</td>
<td>112</td>
<td>108</td>
<td>110</td>
<td>117</td>
<td>117</td>
<td>122</td>
<td>128</td>
</tr>
<tr>
<td>hash/each</td>
<td>100</td>
<td>82</td>
<td>118</td>
<td>125</td>
<td>128</td>
<td>111</td>
<td>117</td>
<td>126</td>
</tr>
<tr>
<td>hash/foreach-sort</td>
<td>100</td>
<td>107</td>
<td>105</td>
<td>114</td>
<td>104</td>
<td>110</td>
<td>112</td>
<td>114</td>
</tr>
<tr>
<td>hash/foreach</td>
<td>100</td>
<td>121</td>
<td>119</td>
<td>124</td>
<td>123</td>
<td>123</td>
<td>108</td>
<td>128</td>
</tr>
<tr>
<td>hash/get</td>
<td>100</td>
<td>140</td>
<td>151</td>
<td>146</td>
<td>151</td>
<td>143</td>
<td>158</td>
<td>153</td>
</tr>
<tr>
<td>hash/set</td>
<td>100</td>
<td>116</td>
<td>140</td>
<td>117</td>
<td>137</td>
<td>121</td>
<td>142</td>
<td>139</td>
</tr>
</tbody>
</table>
### PerlBench results from reini at 2017-08-08 09:29:37

<table>
<thead>
<tr>
<th>Function</th>
<th>S.14</th>
<th>S.22</th>
<th>S.22c</th>
<th>S.24</th>
<th>S.24c</th>
<th>S.26</th>
<th>S.26c</th>
<th>S.27c</th>
</tr>
</thead>
<tbody>
<tr>
<td>hash/copy</td>
<td>100</td>
<td>112</td>
<td>108</td>
<td>110</td>
<td>117</td>
<td>117</td>
<td>122</td>
<td>128</td>
</tr>
<tr>
<td>hash/each</td>
<td>100</td>
<td>82</td>
<td>118</td>
<td>125</td>
<td>128</td>
<td>111</td>
<td>117</td>
<td>126</td>
</tr>
<tr>
<td>hash/foreach-sort</td>
<td>100</td>
<td>107</td>
<td>105</td>
<td>114</td>
<td>104</td>
<td>110</td>
<td>112</td>
<td>114</td>
</tr>
<tr>
<td>hash/foreach</td>
<td>100</td>
<td>121</td>
<td>119</td>
<td>124</td>
<td>123</td>
<td>123</td>
<td>108</td>
<td>128</td>
</tr>
<tr>
<td>hash/get</td>
<td>100</td>
<td>140</td>
<td>151</td>
<td>146</td>
<td>151</td>
<td>143</td>
<td>158</td>
<td>153</td>
</tr>
<tr>
<td>hash/set</td>
<td>100</td>
<td>116</td>
<td>140</td>
<td>117</td>
<td>137</td>
<td>121</td>
<td>142</td>
<td>139</td>
</tr>
<tr>
<td>loop/for-ary</td>
<td>100</td>
<td>118</td>
<td>130</td>
<td>202</td>
<td>213</td>
<td>161</td>
<td>206</td>
<td>246</td>
</tr>
<tr>
<td>loop/for-c</td>
<td>100</td>
<td>103</td>
<td>153</td>
<td>174</td>
<td>156</td>
<td>141</td>
<td>159</td>
<td>148</td>
</tr>
<tr>
<td>loop/for-range-const</td>
<td>100</td>
<td>115</td>
<td>117</td>
<td>141</td>
<td>99</td>
<td>142</td>
<td>92</td>
<td>147</td>
</tr>
<tr>
<td>loop/for-range</td>
<td>100</td>
<td>118</td>
<td>128</td>
<td>146</td>
<td>146</td>
<td>141</td>
<td>96</td>
<td>138</td>
</tr>
<tr>
<td>loop/getline</td>
<td>100</td>
<td>95</td>
<td>100</td>
<td>97</td>
<td>105</td>
<td>99</td>
<td>122</td>
<td>116</td>
</tr>
<tr>
<td>loop/while-my</td>
<td>100</td>
<td>117</td>
<td>142</td>
<td>151</td>
<td>105</td>
<td>152</td>
<td>108</td>
<td>147</td>
</tr>
<tr>
<td>loop/while</td>
<td>100</td>
<td>77</td>
<td>146</td>
<td>151</td>
<td>106</td>
<td>135</td>
<td>115</td>
<td>156</td>
</tr>
<tr>
<td>re/const</td>
<td>100</td>
<td>104</td>
<td>248</td>
<td>240</td>
<td>248</td>
<td>229</td>
<td>262</td>
<td>228</td>
</tr>
<tr>
<td>re/w</td>
<td>100</td>
<td>118</td>
<td>128</td>
<td>122</td>
<td>108</td>
<td>119</td>
<td>142</td>
<td>132</td>
</tr>
<tr>
<td>startup/fewmod</td>
<td>100</td>
<td>86</td>
<td>74</td>
<td>86</td>
<td>72</td>
<td>85</td>
<td>72</td>
<td>71</td>
</tr>
<tr>
<td>startup/lotofsub</td>
<td>100</td>
<td>86</td>
<td>86</td>
<td>86</td>
<td>83</td>
<td>83</td>
<td>82</td>
<td>80</td>
</tr>
<tr>
<td>startup/noprog</td>
<td>100</td>
<td>103</td>
<td>107</td>
<td>104</td>
<td>107</td>
<td>106</td>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td>string/base64</td>
<td>100</td>
<td>113</td>
<td>102</td>
<td>110</td>
<td>106</td>
<td>103</td>
<td>104</td>
<td>102</td>
</tr>
<tr>
<td>string/htmlparser</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>89</td>
<td>94</td>
<td>91</td>
<td>91</td>
<td>94</td>
</tr>
<tr>
<td>string/index-const</td>
<td>100</td>
<td>104</td>
<td>175</td>
<td>188</td>
<td>181</td>
<td>147</td>
<td>190</td>
<td>155</td>
</tr>
<tr>
<td>string/index-var</td>
<td>100</td>
<td>107</td>
<td>146</td>
<td>177</td>
<td>142</td>
<td>292</td>
<td>337</td>
<td>292</td>
</tr>
<tr>
<td>string/ipol</td>
<td>100</td>
<td>102</td>
<td>101</td>
<td>104</td>
<td>101</td>
<td>116</td>
<td>101</td>
<td>115</td>
</tr>
<tr>
<td>string/tr</td>
<td>100</td>
<td>102</td>
<td>102</td>
<td>104</td>
<td>101</td>
<td>118</td>
<td>125</td>
<td>123</td>
</tr>
</tbody>
</table>

**Average**

<table>
<thead>
<tr>
<th></th>
<th>100.0</th>
<th>114.2</th>
<th>123.6</th>
<th>138.2</th>
<th>133.3</th>
<th>140.6</th>
<th>140.5</th>
<th>144.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic mean</td>
<td>100.0</td>
<td>115.4</td>
<td>126.8</td>
<td>142.8</td>
<td>137.4</td>
<td>148.3</td>
<td>148.9</td>
<td>151.3</td>
</tr>
</tbody>
</table>

Higher numbers are better. 200 is twice as fast as 100.
# MEMORY

**Table:**

<table>
<thead>
<tr>
<th>Release</th>
<th>Memory (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.6.2</td>
<td>673653</td>
</tr>
<tr>
<td>5.8.5</td>
<td>753069</td>
</tr>
<tr>
<td>5.8.8</td>
<td>666789</td>
</tr>
<tr>
<td>5.10.1</td>
<td>617331</td>
</tr>
<tr>
<td>5.12.5</td>
<td>656194</td>
</tr>
<tr>
<td>5.14.4</td>
<td>718838</td>
</tr>
<tr>
<td>5.16.3</td>
<td>734030</td>
</tr>
<tr>
<td>5.18.4</td>
<td>800224</td>
</tr>
<tr>
<td>5.20.3</td>
<td>838428</td>
</tr>
<tr>
<td>5.22.2</td>
<td>812375</td>
</tr>
<tr>
<td>5.24.1</td>
<td>815414</td>
</tr>
<tr>
<td>5.26.0</td>
<td>815329</td>
</tr>
<tr>
<td>5.22.4c</td>
<td>596665</td>
</tr>
<tr>
<td>5.24.2c</td>
<td>602570</td>
</tr>
<tr>
<td>5.26.0c</td>
<td>603673</td>
</tr>
</tbody>
</table>

**Diagram:**

A line graph showing memory usage with different versions. The x-axis represents releases, and the y-axis represents memory in bytes.
FEATURES

• support for modern compilers and features
e.g. new gcc 5, clang 3.4, icc intrinsics:
builtin_arith_overflow, builtin_clz, builtin_ctz, builtin_prefetch
FEATURES

 builtin_arith_overflow

i_add:
   add %rax, %rbx
   jo +4

n_add:
   fadd %rax, %rbx
   jo +4

With gcc-5 or clang-3.4 on 64bit new fast compiler intrinsics are used for the add and multiply ops, which check just for the overflow flag and then jump to the promotion to double. This results in smaller and much faster code. The old code which had to compute the results twice, and needed many more branches before to check for IV to UV or NV promotion.

jo - jump if overflow

The recent p5p arithmetic improvements are 48% slower.

8%-25% less instructions and 20% less branches for add and multiply
FEATURES

builtin_clz ("count leading zeros")

hash splits are $10^8$x faster. 100,000,000

time clz: 18 ns
time ceil(log2): 1060744672 ns  fallback if no builtin_clz
time perl5 core: 2904949213 ns
FEATURES

builtin ctz ("count trailing zeros")

allows -DHV_FILL_RATE=90 in DO_HSPLIT
perl5 can only do 100%
FEATURES

• support for modern compilers and features
  -fsanitize=cfi, -flto, -flto=thin, -fstack-check, …

uses LD not CC for link (llvm)

re-introduced C enum’s, removed with 5.6.0
  (perl6 enum later)
TYPES

• Compile-time checks and optimizations
• Faster not slower
• Safer not dynamic test-driven bug-hunting
TYPES

dist/Test-Simple/lib/Test/More.pm:

```perl
sub _whoa ( int $check, str $desc ) {
    if ($check) {
        die <<"WHOA";
    }
    return $ok;
}

_whoa( 1, "No type in _deep_check" );
```

This should never happen! Please contact the author immediately!

WHOA
$ cperl -we 'sub test(int $i) {print $i} test "0"'
Type of arg $i to test must be int (not Str) at -e line 1, at EOF
Execution of -e aborted due to compilation errors.
255
$ cperl -we'sub test(int $i) {print $i} test "0"'
Type of arg $i to test must be int (not Str) at -e line 1, at EOF
Execution of -e aborted due to compilation errors.
255

$ cperl -we'my int @a[2];
@a[0] = "1";'  
Type of scalar assignment to @a must be int (not Str) at -e line 2.
TYPES

$ cperl -we'sub test(int $i) {print $i} test "0"
Type of arg $i to test must be int (not Str) at -e line 1, at EOF
Execution of -e aborted due to compilation errors.

255

$ cperl -we'my int @a[2];
@a[0] = "1";
Type of scalar assignment to @a must be int (not Str) at -e line 2.

$ cperl -we'my int @a[2];
@a[2] = 1;
Array index out of bounds @a[2] at -e line 2.
TYPES

Benefits:

* Modern Perl
* Compile-time safety
* 2x faster with signatures and types
SHAPED ARRAYS

* typed - compile-time checked

* remove run-time out-of-bounds checks

* optimized storage and usage with native types

```
$cperl -we 'my int @a[2];
@a[2] = 1;'
Array index out of bounds @a[2] at -e line 2.
```
SHAPED ARRAYS

Static loop optimizations

* eliminated run-time bounds checks
$ cperl -Dt -e 'my int @a[1]; for my $i (0..$#a) { print $a[$i] }'

EXECUTING...

(-e:0)  enter
(-e:0)  nextstate
(-e:1)  padav(@a)
(-e:1)  nextstate
(-e:1)  pushmark
(-e:1)  const(IV(0))
(-e:1)  padav(@a)
(-e:1)  av2arylen
(-e:1)  enteriter
(-e:1)  iter_lazyiv
(-e:1)  and
(-e:1)  nextstate
(-e:1)  padrange(@a,$i)
(-e:1)  aelem_u
(-e:1)  print
(-e:1)  unstack
(-e:1)  iter_lazyiv
(-e:1)  and
(-e:1)  leaveloop
(-e:1)  leave

_u for unchecked
$ cperl -Dt -e'my int @a[1]; for my $i (0..$#a) { print $a[$i]+1 }'  

EXECUTING...

(-e:0) enter
(-e:0) nextstate
(-e:1) padav(@a)
(-e:1) nextstate
(-e:1) pushmark
(-e:1) int_const(IV(0))
(-e:1) int_padav(@a)
(-e:1) av2arylen
(-e:1) enteriter
(-e:1) iter_lazyiv
(-e:1) and
(-e:1) nextstate
(-e:1) padrange(@a,$i)
(-e:1) int_aelem_u
(-e:1) int_const(IV(1))
(-e:1) int_add
(-e:1) print
(-e:1) unstack
(-e:1) iter_lazyiv
(-e:1) and
(-e:1) leaveloop
(-e:1) leave

native shaped array

typed add (with box)

specialized iter op
UNICODE OPS

use utf8;
{main ⇒ 1};  # FATCOMMA
$obj⇒test;  # -→ ARROW
sort {$a ≡ $b} (2,1);  # ≦ NCMP

1 ≠ 2;  # != NE
1 ≤ 2;  # <= LE
2 ≥ 1;  # >= GE
10 ÷ 2;  # / DIVIDE

my $x=2; @a=($x⁰, 2¹, $x², 2³, $x⁴, $x⁵, $x⁶, $x⁷, $x⁸, $x⁹);

(2⁵)⁵
2⁵⁵
UNICODE SECURITY

```
use utf8;
my $Г = 1;
if ($Г) { warn; }
```

Greek
Cyrillic

Invalid script Cyrillic in identifier Г) { warn } for U+0413. Have Greek at -e line 1.
UNICODE SECURITY

$ perl -e'use utf8; $ab == 2;'

$ cperl -e'use utf8; $ab == 2;'
Unrecognized character \x{3164}; marked by <-- HERE after e utf8; $a<-- HERE near column 13 at -e line 1.

Hangul Filler unicode bug
UNICODE SECURITY

Identifier normalization

```
$ perl -C -Mutf8 -we'
sub café { "slurf" }
sub café { "drink" }
print café;'  
slurf

$ cperl -C -Mutf8 -we'
sub café { "slurf" }
sub café { "drink" }
print café;'  
Subroutine café redefined at -e line 3.
drink
```
UNICODE SECURITY

Identifier normalization

$ perl -C -Mutf8 -we'
sub café { "slurf" }
sub café { "drink" }
print café;
slurf

$cperl -C -Mutf8 -we'
sub café { "slurf" }
sub café { "drink" }
print café;
Subroutine café redefined at -e line 3.
drink

<c, a, f, e, U+0301>
<c, a, f, U+00E9>

normalized to <c, a, f, U+00E9>
strict 'names'

```perl
perl -C -Mutf8 -e'
use strict; no strict "refs";
${"\xc3\x28"}
'

perl -C -Mutf8 -e'
use strict; no strict "refs";
${"\xc3\x28"}
'
Invalid identifier "\303" while "strict names" in use at -e line 4.

Enforce parsable identifiers
MODERN TOOLCHAIN

p5p is not able to maintain the core modules

Critical fixes in:
• Storable
• ExtUtils-Constant, threads, …

Modernized:
• Test-Simple, Pod-Simple, Pod-HTML, PathTools, bigint/bignum/bigrat, c2ph

Fast XS in core:
• Cpanel::JSON::XS, YAML::XS, strict, attributes, DynaLoader, Config, XSLoader, Devel::NYTProf
Native support for the ‘c’ version suffix, builtin XS modules, cperl extensions, improved perf. + safety on most other toolchain modules (CPAN, YAML, …)

```bash
$ cperl -e'print $^V'
v5.27.1c

$^V =~ /c$/             # cperl detection
$Config{usecperl}       # cperl detection
```
MODERN TOOLCHAIN

**Test2** is currently being blocked.

It is a good idea to improve streaming of subtests, but only if
* it’s not >20% slower,
* it would be optional, not mandatory.

It is in work to get into acceptable performance by modernization of the API (faster via signatures), but still too slow.

Previously you could at least trust CPAN, not p5p, but recently even CPAN eroded:
Module::Build, YAML, Moose, Scalar::Util, EUMM, Test::Simple
Recommended was **Mouse** or use base/fields with pseudohashes.

cperl classes, roles look like perl6, and behave like Mouse. Definitely no Moxie %HAS, still %FIELDS.

The type system is fast and integrated. Several perl6 design bugs are fixed.

But no safe structural typing, no sound types, very weak type inference.
class MyPoint does MyBasepoint {
    has num $x = 0.0;
    has num $y = 0.0;
    method dist (MyPoint $p) {
        sqrt( ($p->x - $x)^2 + ($p->y - $y)^2 )
    }
    multi dist (My3DPoint $a, MyPoint $b) {
        sqrt( ($a->x - $b->x)^2 + ($a->y - $b->y)^2 + $a->z^2 )
    }
    multi dist (My3DPoint $a, MyPoint $b) :after {
        log "dist My3DPoint $a, MyPoint $b"
    }
}

# ditto for role and does
OBJECTS

Built-in ffi (NativeCall is taken with worse syntax)

```perl
class MyPoint :native {
    has int $x;
    has int $y;
}
extern sub gtk2_draw (MyPoint $p);
```

Sugar for the also supported perl6 syntax:

```perl
class MyPoint is repr(CStruct) {
    has int $x;
    has int $y;
}
sub gtk2_draw (MyPoint $p) :native;
```
ATTRIBUTES

Lots of builtin attributes

• :const for all data types
• :pure for functions
• coretypes for signatures and as function return type
• user-types for signatures and as function return type
• safe run-time attribute values
  (no Attribute::Handler eval security risk)
cperl uses attributes for everything perl6 does with traits.

```perl
sub add(int32, int32) returns int32 is native("calculator") { * }
sub add(int32 $i, int32 $j) :int32 :native("calculator");
```
ATTRIBUTES

cperl fixes perl5 attributes, and improves on perl6 traits.

Even perl6 has the design bug, that trait arguments are compile-time only.

```perl
my $libname = "mysqlclient";
$libname = "cygmysqlclient-18.dll" if $^O eq 'cygwin';

sub mysql_init( OpaquePointer $mysql_client ) :OpaquePointer :native($libname);
```

The FFI needs application logic at run-time to find a DLL.

Defer the attributes->import call to run-time, as with my $var :attr; declarations.
BUGFIXES

Worst perl5 bugs
Worst perl5 bugs

Denial of Service

It's trivial to DoS a perl5 system.

```perl
$a[9223372036854]=0;

@a=(0..4294967296);
```

Examples for a 64bit system, but also trivial on 32bit.
It creates a huge array or hash, which runs out of memory in the VMM subsystem which eventually kills the process. cperl dies with "Too many elements", here even at compile-time.
BUGFIXES

Worst perl5 bugs

No Hash Security

Only security theatre with “secure hash functions” and slower collision resolution KEY_PERTURBRANDOM. Secure hash functions for hash tables do not exist! Hash function security starts with 256 bits, we have 32.
Only collision resolution can protect a hash table properly. Even siphash is brute foricable (just 2x slower)

```perl
$ PERL_HASH_SEED_DEBUG=1 perl -e1
  HASH_FUNCTION = ONE_AT_A_TIME_HARD HASH_SEED = 0x5d5e02649c4f1e25
  PERTURB_KEYS = 1 (RANDOM)
```

```perl
$ PERL_HASH_SEED_DEBUG=1 cperl -e1
  HASH_FUNCTION = FNV1A HASH_SEED = <hidden> PERTURB_KEYS = 0 (TOP)
```
Worst perl5 bugs

Silent integer overwraps

```perl
@a=(0,1); print $a[~1] => 0
```

`~1` is essentially `(UV)-2` or `0xffff_fffe`.

```perl
@a=(1); print $a[18446744073709551615] => 1
```

Silent overwrap of 18446744073709551615 to -1.

```
$ cperl -e '@a=(1); print $a[18446744073709551615]'
Too many elements at -e line 1.
```
Worst perl5 bugs

Inconsistencies with over-long data (I32 - U64)

* Silent hash or array overflows.
* `chop/chomp` only works on half of overlarge arrays.
* Hash keys > I32 silently truncated, not converted to SV or error.
* Hash insertion allows >I32, but iteration only I32.
* Storable cannot handle >I32
* ~"a"x2G complement silently truncated.
* `smartmatch Array - CodeRef` processes only over half the array elements.
* Regex match group of >2GB string len.
* Repeat count >2GB and don't silently cap it at IV_MAX. Which was at least better than silent wrap around.

Most of them security relevant.
BUGFIXES

Worst perl5 bugs

Names

* Silently introduces binary names, without any protection. Called "harmonization".
* Insecure unicode names
* Overlong names (silently truncated at I32_MAX)
* Same for file, keyword, mro, stash names.

More symbol table nonsense:
autovivification was for decades considered a bug, eventually fixed. Now it’s declared a feature. Searching a name will create the name by perl5 magic side effect. Will be fixed in cperl eventually, with the symbol table rewrite for a single flat table.
Worst perl5 bugs

Insecure taint mode

Bugzilla is plagued for years by the existing taint loopholes. You cannot trust perl5 tainting. They should just switch over to cperl to be safe again.

See `perldoc perlsec` vs `cperldoc perlsec`
Hash keys keep the tainted bit, regex capturing is re ‘taint’ safe.
BUGFIXES

Worst perl5 bugs

**Lexical my $$_**

An important feature for nested loops, iterators, given/when rules, and much faster than global $$_ lookup.

Fixed in cperl, removed as “unfixable” in perl5.
Worst perl5 bugs

for (qw(a b c)) {}

p5p thinks that this new feature is better than the old

for qw(a b c) {}

cperl allows for qw(a b c) {} again.

It’s only ~10 lines in the parser.
Worst perl5 bugs

The COW system

Copy-on-write vs Copy-on-grow

perl5 does nothing of those. It would be great to have at least one these.

B::C supports both, but has to disable perl5 COW for increasing memory usage by 20% since 5.18.

It’s also unmaintainable. I’ve tried to rewrite it twice.
Worst perl5 bugs

The double readonly system, SVf_PROTECT

Was trivial to fix in cperl. No need to check for 2 bits for readonly-ness, and compile-time optimizations harmonize with the run-time.

The root cause in perl5 is still broken though.
IN WORK
IN WORK

A lot!

Unlike perl5 cperl is actively developed
A lot!

Unlike perl5 cperl is actively developed

And their unofficial TODO list is frightening:

* refcounted stack
* hash table via vtable (one more indirection)
* cow rewrite
IN WORK

• class/role/multi/method/has
IN WORK

- class/role/multi/method/has
- ffi
IN WORK

• class/role/multi/method/has
• ffi
• native types
IN WORK

• class/role/multi/method/has
• ffi
• native types
• inline subs
IN WORK

- class/role/multi/method/has
- ffi
- native types
- inline subs
- hash table rewrite (open addressing)
IN WORK

• class/role/multi/method/has
• ffi
• native types
• inline subs
• hash table rewrite (open addressing)
• no miniperl

...
IN WORK

• unexec
IN WORK

• unexec
• lazy parsing
IN WORK

- unexec
- lazy parsing
- llvm jit
IN WORK

- unexec
- lazy parsing
- llvm jit
- flat symbol table (gvlinear)
IN WORK

- unexec
- lazy parsing
- llvm jit
- flat symbol table (gvlinear)
- warnings and Carp as builtin XS
IN WORK

- unexec
- lazy parsing
- llvm jit
- flat symbol table (gvlinear)
- warnings and Carp as builtin XS
- unicode tables as XS
IN WORK

• unexec
• lazy parsing
• llvm jit
• flat symbol table (gvlinear)
• warnings and Carp as builtin XS
• unicode tables as XS
• fast regex compiler (hyperscan + pcre2-jit)
IN WORK

- unexec
- lazy parsing
- llvm jit
- flat symbol table (gvlinear)
- warnings and Carp as builtin XS
- unicode tables as XS
- fast regex compiler (hyperscan + pcre2-jit)

… ~90 feature branches
POLICIES

- No worst practices as in p5p
POLICIES

• No worst practices as in p5p
• Best practices as in parrot, perl6
POLICIES

• No worst practices as in p5p
• Best practices as in parrot, perl6
• New branch rebase policy to keep all work branches up to date
POLICIES

• No worst practices as in p5p
• Best practices as in parrot, perl6
• New branch rebase policy to keep all work branches up to date
• No tragedy of the commons by failing to pick votes for backports
POLICIES

• No worst practices as in p5p
• Best practices as in parrot, perl6
• New branch rebase policy to keep all work branches up to date
• No tragedy of the commons by failing to pick votes for backports
• No death by code of conduct
POLICIES

• No worst practices as in p5p
• Best practices as in parrot, perl6
• New branch rebase policy to keep all work branches up to date
• No tragedy of the commons by failing to pick votes for backports
• No death by code of conduct
• No promotion of bad behavior as in p5p or YAPC
QUESTIONS