What's New in Perl v5.0?

Tom Christiansen

<tchrist@cs.colorado.edu>

"There's more than one way to do it!"

Overview

- General Information
- Incompatibilities
- Interpreter Rewrite
- Fewer Silent Surprises
- Static Scoping
- References and Dereferencing
- Nested Data Structures
- Structures and Classes
- BEGIN/END Routines
- GUI Perl
- Development Tools, Libraries, Documentation
- Miscellanea
- Desiderata

Why, When, Where, Who, What?

CAVEAT LECTOR — no guarantees here!

- Why: because we need the "what".
- When: alpha now, beta *maybe* around New Years, "production" *maybe* around Easter.
- Where: alpha3 out from ftp.netlabs.com in /pub/outgoing/perl5.0 .
- Who: Larry Wall (kernel) in his "spare" time, and Tom Christiansen (non-kernel) when not lecturing; possibly others, like Jon Biggar for guiperl.
- What: see the rest of the presentation. :-)

Incompatibilities

Very few Perl v4 scripts will have any problems. Here are some, however:

- `s'$lhs'$rhs'` **and** `splice()` **now work as documented.**
- `shift @foo + 20`
- `open FOO || die` **needs parens**
- **Formats args now evaluated in list context**
- **goto into a block optimized away.**
- **No more vars with whitespace names.**
- **Some error messages will be different**
- **Caller can return false if no caller**
- `m//g` **attaches state to searched string**
- **functions named BEGIN() and END()**

Interpreter Rewrite

- **Nearly total rewrite of interpreter; yacc grammar much simpler.**
- **Listops and subs unified**

```
split /:/, $pwent;
afunc @list, @list2;
```

- **Huge switch statements now gone:**

```
run() {
     while ( op = (*op->op_ppaddr)() ) ;
}
```

- **No global variables; embeddable.**
- **Faster to execute (usually).**
- **Slower to compile.**

Silent Surprises

- The -w flag now drastically more useful.
- Catches you when you use scalar for list or string for number, or vice versa:
  - `$a[1,2]`
  - `@x[2]`
  - `$a < 'red'`
- Forgetting octal 0 on umask, mkdir, chmod.
- Extra newlines on filenames.
- `next`, `last`, or `redo` out of static scope.
- Precedence problems and more.

Static scoping

- **`my` does static scoping:**

```
my $foo = shift;
```

- **Blocks can have "no" package, allowing compile-time errors for accidental variable references:**

```
sub func {
    my $foo = shift;
    package;
    $main'count += $foo;
    $bar = 1;  # compile error
}
```

References

- All scalar values have reference counts on them now. Nothing is freed until refcounts go to zero.
- Compiling with -O makes a big difference.
- References are typed, being a reference to either a scalar, list, table, function, another reference, or a user-defined class.
- Typeless names can be used for references, providing most of the dangerous `*foo` type glob functionality safely.

Getting a Reference

- **Use a backslash to get a reference to something:**

```
$ref = \$scalar;
$ref = \@list;
$ref = \%table;
$ref = \&function;
$refref = \$ref;
```

Prefix Dereferencing

- **Where C uses a `*`, Perl uses one of its four typed dereferencing operators, `$ @ % &`:**

```
print $$ref;
push (@$ref, @newelts);
keys %$ref;
&{ $table{$cmd} } ($args);
```

- **These can stack:**

```
$ref = \$foo;
$refref = \$ref;
print $$$refref;
```

Passing to Functions

- Easy to pass references without dangers of `*foo` typeglobs:

```
&myfunc(\@x, \%t, \&nuther_func);

sub myfunc {
    my $lref = shift;
    my $tref = shift;
    my $fref = shift;
}
```

- `ref()` function returns the type of reference:

```
die "bad ref" unless ref($this) eq "CODE";
```

Postfix Dereferencing

- Where C uses a ->, Perl does, too! Useful for nested lists or tables, or for classes.

```
$ref = \@list;
$ref->[2] = 7;
$ref[2] = 8; # WARNING: list, not deref
```

- You can use lots of these:

```
$a = \%b;
$a->{$user}->[$i]->{"keyval"}->[8] = 1;
$a->{$user}[$i]{"keyval"}[8] = 1;
$b{$user}->[$i]->{"keyval"}->[8] = 1;
$b{$user}[$i]{"keyval"}[8] = 1;
```

Nested Lists

- **For a nested list, use [ ] for a reference to a list. Note that these are different:**

```
@x = ( 1, 3, [5, 4], 2 );
$xref = [ 1, 3, [5, 4], 2 ];

$x[2][1]       = 4;
$xref->[2][1] = 4;
```

Nested Tables

- **For a nested list, use { }.**

```
%colors = (
  RED  => { CRIMSON => 1, SCARLET => 2 },
  BLUE => { AZURE   => 1, INDIGO  => 2 },
);

$colors{RED}->{CRIMSON} = 1;
$colors{RED}{CRIMSON}   = 1;
```

Combining Lists and Tables

- **These can be mixed and matched:**

```
$BB = {    # set up black book table
     John =>  [ Pat,   Mary, Alice ],
     Paul =>  [ Sally, Jill, Tracy ],
     Mark =>  [ Ann,   John, Dawn  ],
};

if ($BB->{John}->[2] eq "Alice") {}
if ($BB->{John}[2] eq "Alice")   {}
```

How to Model structs?

- **Probably the easiest way to model structs is via an associative array:**

```
$utime =
    $u->{ru_utime}->{tv_sec} +
    $u->{ru_utime}->{tv_usec} / 1_000_000;
```

- **More "type safe" and interesting effects can be achieved using object methods; good for *c2ph*:**

```
$utime =
    $u->ru_utime->tv_sec +
    $u->ru_utime->tv_usec / 1_000_000;
```

- **Maybe `wantlvalue()` function:**

```
    ${$sb->st_mode} &= 07777;
```

- **A library will be provided to make it easy to create these struct-like class objects.**

Perl with Classes

- What does this do? :-)

```
perl -le 'print ++($x = perl)'
```

- An object is just a reference that has been "approved" by some package. Its methods are just subs in that package.
- You can call object methods on a reference:

```
    $r -> bump($arg1,$arg2);
```

- Familiar alternative syntax works like this:

```
    bump $r $arg1,$arg2;
```

More on Classes

- The first argument to a method is the object reference:

```
     sub bump { my $this = shift }
```

- Methods are inherited according to the @ISA list:

```
     package MYOB;
     @ISA = (BaseOb, HisOb, HerOb);
```

- To get a reference to be ‘‘owned’’ by a package, it must be ‘‘blessed’’ by that package (in the corporate, not the ecclesiastical sense :-)

```
     sub new {
          my $ref = bless {}; # new table
          $ref->{"__Start"} = &Start();
          $ObCount++; return $ref;
     }
```

Constructors, Destructors

- **Global constructors are the package init code.**
- **Global destructors are the package `END` routine.**
- **Object destructors are the package `DESTROY` routine.**
- **Object constructors must be user-defined.**

```
$ref = &MyOb'new();
$ref = new MyOb;    # not yet

$dupref = $ref;

$saveob = $ref->copy();
$saveob = copy $ref;
```

guiperl

- Working guiperl for internal v4.5 using OI bindings from ParcPlace.
- Need bindings for Motif, Xt, maybe even a tkperl. Problem: No one has time for this right now.
- Biggest obstacle is selecting a usable Perl5 struct model, then fixing up existing guiperl to use it.
- FYI: Many people are very happy with WAFE.

Development Tools

- debugger updates and enhancements
- dynamic loading package
- a profiler (or two)
- a cross referencer
- tags program
- autoloading tools
- *h2ph* and *c2ph* updates; *h2ph* on install
- better pus tools for C linkage help
- Man pages for everything on this page

Possible Debugger Enhancements

- info command
- whatis command
- printing: pretty, recurse, object
- sh -x emulator
- make able to save state; rerun
- arrange to execute `DB'END()` last and catch it
- allow users to set breakpoints from their program
- make making aliases trivial, intuitive
- add the dot command standard
- make `list` behave more like gdb
- watch var; watch line
- pager?
- vi/xterm multiwindow support

Possible New Libraries

- option mangling functions
- autoloading (from disk or self)
- classlike libs for anything returning a list `stat($file)->st_mode)` or `getpwnam($me)->pw_passwd`
- `trusted($file)` (for dot files, etc; recursive)
- realpath()
- exception based versions of syscalls?
- S-expr read and write
- *cut* (1)-like frontend for unpack, "1-3 4-10 11-20" meaning "A3 A7 A10"
- `gensym()`
- `pushstream()` for pagers?

More Possible New Libraries

- import/export
- atexit()
- date manipulation (getdate, julian, etc)
- stack inspectors
- `sysopen()` unless builtin
- a function for doing s/foo/bar/I
- a GNUish `readline()` function
- a comma-fyer
- &sub() function
- library class: sets, ordered collections
- posix/ansi stuff??? stftime(), sigblock(), and friends
- ftp library, maybe nntp

Documentation

- Explain C and C++ linkage mechanisms.
- All tools get own man pages in 1pl, and sysadmin ones in 8pl.
- All libraries get their own man pages, e.g. *atexit* (3pl)
- Break *perl* (1) up so each built-in has its own man page e.g. *unlink* (2pl).
- Make man pages for concept areas, like ipc, regexps, dynamic loading in section 7pl.

Miscellanea

- Numerous cleanups and bug fixes
- q {} nestability
- Subs without &'s
- Subs as list ops (no parens)
- Mnemonics for $<punctuation> vars
- File handles as objects
- m//g state attached to var, not re

Desiderata

- **Would like VHL switch statement, `in` operator, built-in globbing, non-greedy REs, binding variables to packages for generalized `dbmopen()`, BSD *db* (3) package, and generalized iterators:**

```
while ( <$obref> ) { }
while ( $_ = $obref->iterate() ) { }
foreach $ob ( <$obref> ) { }
```

- **Things not essential to semantics of language are being delayed:**
  - **Saving of semi-compiled p-code.**
  - **Compiling all the way to C, and thence to machine code.**
  - **Direct multi-threading support with scheduling.**