NAME

perltodo - Perl TO-DO List

DESCRIPTION

This is a list of wishes for Perl. The tasks we think are smaller or easier are listed first. Anyone is welcome to work on any of these, but it's a good idea to first contact *perl5-porters@perl.org* to avoid duplication of effort. By all means contact a pumpking privately first if you prefer.

Whilst patches to make the list shorter are most welcome, ideas to add to the list are also encouraged. Check the perl5-porters archives for past ideas, and any discussion about them. One set of archives may be found at:

```
http://www.xray.mpe.mpg.de/mailing-lists/perl5-porters/
```

What can we offer you in return? Fame, fortune, and everlasting glory? Maybe not, but if your patch is incorporated, then we'll add your name to the *AUTHORS* file, which ships in the official distribution. How many other programming languages offer you 1 line of immortality?

The roadmap to 5.10

The roadmap to 5.10 envisages feature based releases, as various items in this TODO are completed.

Needed for a 5.9.4 release

- Review assertions. Review syntax to combine assertions. Assertions could take advantage of the lexical pragmas work. *What hooks would assertions need?*

Needed for a 5.9.5 release

* Implement *_ prototype character*

* Implement *state variables*

Needed for a 5.9.6 release

Stabilisation. If all goes well, this will be the equivalent of a 5.10-beta.

Tasks that only need Perl knowledge

common test code for timed bail out

Write portable self destruct code for tests to stop them burning CPU in infinite loops. This needs to avoid using alarm, as some of the tests are testing alarm/sleep or timers.

POD -> HTML conversion in the core still sucks

Which is crazy given just how simple POD purports to be, and how simple HTML can be. It's not actually *as* simple as it sounds, particularly with the flexibility POD allows for `=item`, but it would be good to improve the visual appeal of the HTML generated, and to avoid it having any validation errors. See also *make HTML install work*, as the layout of installation tree is needed to improve the cross-linking.

The addition of `Pod::Simple` and its related modules may make this task easier to complete.

Parallel testing

The core regression test suite is getting ever more comprehensive, which has the side effect that it takes longer to run. This isn't so good. Investigate whether it would be feasible to give the harness script the **option** of running sets of tests in parallel. This would be useful for tests in *t/op/*.t* and *t/uni/*.t* and maybe some sets of tests in *lib/*.

Questions to answer

1. How does screen layout work when you're running more than one test?

2 How does the caller of test specify how many tests to run in parallel?

3 How do setup/teardown tests identify themselves?

Pugs already does parallel testing - can their approach be re-used?

Make Schwern poorer

We should have for everything. When all the core's modules are tested, Schwern has promised to donate to $500 to TPF. We may need volunteers to hold him upside down and shake vigorously in order to actually extract the cash.

See *t/lib/1_compile.t* for the 3 remaining modules that need tests.

Improve the coverage of the core tests

Use Devel::Cover to ascertain the core's test coverage, then add tests that are currently missing.

test B

A full test suite for the B module would be nice.

A decent benchmark

`perlbench` seems impervious to any recent changes made to the perl core. It would be useful to have a reasonable general benchmarking suite that roughly represented what current perl programs do, and measurably reported whether tweaks to the core improve, degrade or don't really affect performance, to guide people attempting to optimise the guts of perl. Gisle would welcome new tests for perlbench.

fix tainting bugs

Fix the bugs revealed by running the test suite with the `-t` switch (via `make test.taintwarn`).

Dual life everything

As part of the "dists" plan, anything that doesn't belong in the smallest perl distribution needs to be dual lifed. Anything else can be too. Figure out what changes would be needed to package that module and its tests up for CPAN, and do so. Test it with older perl releases, and fix the problems you find.

Improving threads::shared

Investigate whether `threads::shared` could share aggregates properly with only Perl level changes to shared.pm

POSIX memory footprint

Ilya observed that use POSIX; eats memory like there's no tomorrow, and at various times worked to cut it down. There is probably still fat to cut out - for example POSIX passes Exporter some very memory hungry data structures.

Tasks that need a little sysadmin-type knowledge

Or if you prefer, tasks that you would learn from, and broaden your skills base...

Relocatable perl

The C level patches needed to create a relocatable perl binary are done, as is the work on *Config.pm*. All that's left to do is the `Configure` tweaking to let people specify how they want to do the install.

make HTML install work

There is an `installhtml` target in the Makefile. It's marked as "experimental". It would be good to get this tested, make it work reliably, and remove the "experimental" tag. This would include

1 Checking that cross linking between various parts of the documentation works. In particular

that links work between the modules (files with POD in *lib/*) and the core documentation (files in *pod/*)

2 Work out how to split `perlfunc` into chunks, preferably one per function group, preferably with general case code that could be used elsewhere. Challenges here are correctly identifying the groups of functions that go together, and making the right named external cross-links point to the right page. Things to be aware of are `-X`, groups such as `getpwnam` to `endservent`, two or more `=items` giving the different parameter lists, such as

```
    =item substr EXPR,OFFSET,LENGTH,REPLACEMENT

    =item substr EXPR,OFFSET,LENGTH

    =item substr EXPR,OFFSET
```

and different parameter lists having different meanings. (eg `select`)

compressed man pages

Be able to install them. This would probably need a configure test to see how the system does compressed man pages (same directory/different directory? same filename/different filename), as well as tweaking the *installman* script to compress as necessary.

Add a code coverage target to the Makefile

Make it easy for anyone to run Devel::Cover on the core's tests. The steps to do this manually are roughly

- do a normal `Configure`, but include Devel::Cover as a module to install (see *INSTALL* for how to do this)

-
  ```
      make perl
  ```

-
  ```
      cd t; HARNESS_PERL_SWITCHES=-MDevel::Cover ./perl -I../lib
  harness
  ```
- Process the resulting Devel::Cover database

This just give you the coverage of the *.pm*s. To also get the C level coverage you need to

- Additionally tell `Configure` to use the appropriate C compiler flags for `gcov`

-
  ```
      make perl.gcov
  ```

  (instead of `make perl`)
- After running the tests run `gcov` to generate all the *.gcov* files. (Including down in the subdirectories of *ext/*
- (From the top level perl directory) run `gcov2perl` on all the `.gcov` files to get their stats into the cover_db directory.
- Then process the Devel::Cover database

It would be good to add a single switch to `Configure` to specify that you wanted to perform perl level coverage, and another to specify C level coverage, and have `Configure` and the *Makefile* do all the right things automatically.

Make Config.pm cope with differences between build and installed perl

Quite often vendors ship a perl binary compiled with their (pay-for) compilers. People install a free compiler, such as gcc. To work out how to build extensions, Perl interrogates `%Config`, so in this situation `%Config` describes compilers that aren't there, and extension building fails. This forces people into choosing between re-compiling perl themselves using the compiler they have, or only using modules that the vendor ships.

It would be good to find a way teach `Config.pm` about the installation setup, possibly involving probing at install time or later, so that the `%Config` in a binary distribution better describes the installed machine, when the installed machine differs from the build machine in some significant way.

make parallel builds work

Currently parallel builds (such as `make -j3`) don't work reliably. We believe that this is due to incomplete dependency specification in the *Makefile*. It would be good if someone were able to track down the causes of these problems, so that parallel builds worked properly.

linker specification files

Some platforms mandate that you provide a list of a shared library's external symbols to the linker, so the core already has the infrastructure in place to do this for generating shared perl libraries. My understanding is that the GNU toolchain can accept an optional linker specification file, and restrict visibility just to symbols declared in that file. It would be good to extend *makedef.pl* to support this format, and to provide a means within `Configure` to enable it. This would allow Unix users to test that the export list is correct, and to build a perl that does not pollute the global namespace with private symbols.

Tasks that need a little C knowledge

These tasks would need a little C knowledge, but don't need any specific background or experience with XS, or how the Perl interpreter works

Make it clear from -v if this is the exact official release

Currently perl from `p4/rsync` ships with a *patchlevel.h* file that usually defines one local patch, of the form "MAINT12345" or "RC1". The output of perl -v doesn't report that a perl isn't an official release, and this information can get lost in bugs reports. Because of this, the minor version isn't bumped up until RC time, to minimise the possibility of versions of perl escaping that believe themselves to be newer than they actually are.

It would be useful to find an elegant way to have the "this is an interim maintenance release" or "this is a release candidate" in the terse -v output, and have it so that it's easy for the pumpking to remove this just as the release tarball is rolled up. This way the version pulled out of rsync would always say "I'm a development release" and it would be safe to bump the reported minor version as soon as a release ships, which would aid perl developers.

This task is really about thinking of an elegant way to arrange the C source such that it's trivial for the Pumpking to flag "this is an official release" when making a tarball, yet leave the default source saying "I'm not the official release".

Tidy up global variables

There's a note in *intrpvar.h*

```
/* These two variables are needed to preserve 5.8.x bincompat because
   we can't change function prototypes of two exported functions.
   Probably should be taken out of blead soon, and relevant prototypes
   changed.  */
```

So doing this, and removing any of the unused variables still present would be good.

Ordering of "global" variables.

thrdvar.h and *intrpvarh* define the "global" variables that need to be per-thread under ithreads, where the variables are actually elements in a structure. As C dictates, the variables must be laid out in order of declaration. There is a comment `/* Important ones in the first cache line` `(if alignment is done right) */` which implies that at some point in the past the ordering was carefully chosen (at least in part). However, it's clear that the ordering is less than perfect, as currently there are things such as 7 `bool`s in a row, then something typically requiring 4 byte alignment, and then an odd `bool` later on. (`bool`s are typically defined as `char`s). So it would be good for someone to review the ordering of the variables, to see how much alignment padding can be removed.

bincompat functions

There are lots of functions which are retained for binary compatibility. Clean these up. Move them to mathom.c, and don't compile for blead?

am I hot or not?

The idea of *pp_hot.c* is that it contains the *hot* ops, the ops that are most commonly used. The idea is that by grouping them, their object code will be adjacent in the executable, so they have a greater chance of already being in the CPU cache (or swapped in) due to being near another op already in use.

Except that it's not clear if these really are the most commonly used ops. So anyone feeling like exercising their skill with coverage and profiling tools might want to determine what ops *really* are the most commonly used. And in turn suggest evictions and promotions to achieve a better *pp_hot.c*.

emulate the per-thread memory pool on Unix

For Windows, ithreads allocates memory for each thread from a separate pool, which it discards at thread exit. It also checks that memory is free()d to the correct pool. Neither check is done on Unix, so code developed there won't be subject to such strictures, so can harbour bugs that only show up when the code reaches Windows.

It would be good to be able to optionally emulate the Window pool system on Unix, to let developers who only have access to Unix, or want to use Unix-specific debugging tools, check for these problems. To do this would involve figuring out how the `PerlMem_*` macros wrap `malloc()` access, and providing a layer that records/checks the identity of the thread making the call, and recording all the memory allocated by each thread via this API so that it can be summarily free()d at thread exit. One implementation idea would be to increase the size of allocation, and store the `my_perl` pointer (to identify the thread) at the start, along with pointers to make a linked list of blocks for this thread. To avoid alignment problems it would be necessary to do something like

```
  union memory_header_padded {
    struct memory_header {
      void *thread_id;   /* For my_perl */
      void *next;        /* Pointer to next block for this thread */
    } data;
    long double padding; /* whatever type has maximal alignment constraint
*/
  };
```

although `long double` might not be the only type to add to the padding union.

reduce duplication in sv_setsv_flags

`Perl_sv_setsv_flags` has a comment `/* There's a lot of redundancy below but we're going for speed here */`

Whilst this was true 10 years ago, the growing disparity between RAM and CPU speeds mean that the trade offs have changed. In addition, the duplicate code adds to the maintenance burden. It would be good to see how much of the redundancy can be pruned, particular in the less common paths. (Profiling tools at the ready...). For example, why does the test for "Can't redefine active sort subroutine" need to occur in two places?

Tasks that need a knowledge of XS

These tasks would need C knowledge, and roughly the level of knowledge of the perl API that comes from writing modules that use XS to interface to C.

IPv6

Clean this up. Check everything in core works

shrink GVs, CVs

By removing unused elements and careful re-ordering, the structures for `AV`s and `HV`s have recently been shrunk considerably. It's probable that the same approach would find savings in `GV`s and `CV`s, if not all the other larger-than-`PVMG` types.

merge Perl_sv_2[inpu]v

There's a lot of code shared between `Perl_sv_2iv_flags`, `Perl_sv_2uv_flags`, `Perl_sv_2nv`, and `Perl_sv_2pv_flags`. It would be interesting to see if some of it can be merged into common shared static functions. In particular, `Perl_sv_2uv_flags` started out as a cut&paste from `Perl_sv_2iv_flags` around 5.005_50 time, and it may be possible to replace both with a single function that returns a value or union which is split out by the macros in *sv.h*

UTF8 caching code

The string position/offset cache is not optional. It should be.

Implicit Latin 1 => Unicode translation

Conversions from byte strings to UTF-8 currently map high bit characters to Unicode without translation (or, depending on how you look at it, by implicitly assuming that the byte strings are in Latin-1). As perl assumes the C locale by default, upgrading a string to UTF-8 may change the meaning of its contents regarding character classes, case mapping, etc. This should probably emit a warning (at least).

This task is incremental - even a little bit of work on it will help.

autovivification

Make all autovivification consistent w.r.t LVALUE/RVALUE and strict/no strict;

This task is incremental - even a little bit of work on it will help.

Unicode in Filenames

chdir, chmod, chown, chroot, exec, glob, link, lstat, mkdir, open, opendir, qx, readdir, readlink, rename, rmdir, stat, symlink, sysopen, system, truncate, unlink, utime, -X. All these could potentially accept Unicode filenames either as input or output (and in the case of system and qx Unicode in general, as input or output to/from the shell). Whether a filesystem - an operating system pair understands Unicode in filenames varies.

Known combinations that have some level of understanding include Microsoft NTFS, Apple HFS+ (In Mac OS 9 and X) and Apple UFS (in Mac OS X), NFS v4 is rumored to be Unicode, and of course Plan 9. How to create Unicode filenames, what forms of Unicode are accepted and used (UCS-2, UTF-16, UTF-8), what (if any) is the normalization form used, and so on, varies. Finding the right level of interfacing to Perl requires some thought. Remember that an OS does not implicate a filesystem.

(The Windows -C command flag "wide API support" has been at least temporarily retired in 5.8.1, and the -C has been repurposed, see *perlrun*.)

Unicode in %ENV

Currently the %ENV entries are always byte strings.

use less 'memory'

Investigate trade offs to switch out perl's choices on memory usage. Particularly perl should be able to give memory back.

This task is incremental - even a little bit of work on it will help.

Re-implement :unique in a way that is actually thread-safe

The old implementation made bad assumptions on several levels. A good 90% solution might be just to make `:unique` work to share the string buffer of SvPVs. That way large constant strings can be shared between ithreads, such as the configuration information in *Config*.

Make tainting consistent

Tainting would be easier to use if it didn't take documented shortcuts and allow taint to "leak" everywhere within an expression.

readpipe(LIST)

system() accepts a LIST syntax (and a PROGRAM LIST syntax) to avoid running a shell. readpipe() (the function behind qx//) could be similarly extended.

Tasks that need a knowledge of the interpreter

These tasks would need C knowledge, and knowledge of how the interpreter works, or a willingness to learn.

lexical pragmas

Document the new support for lexical pragmas in 5.9.3 and how %^H works. Maybe `re`, `encoding`, maybe other pragmas could be made lexical.

Attach/detach debugger from running program

The old perltodo notes "With `gdb`, you can attach the debugger to a running program if you pass the process ID. It would be good to do this with the Perl debugger on a running Perl program, although I'm not sure how it would be done." ssh and screen do this with named pipes in /tmp. Maybe we can too.

Constant folding

The peephole optimiser should trap errors during constant folding, and give up on the folding, rather than bailing out at compile time. It is quite possible that the unfoldable constant is in unreachable code, eg something akin to `$a = 0/0 if 0;`

LVALUE functions for lists

The old perltodo notes that lvalue functions don't work for list or hash slices. This would be good to fix.

LVALUE functions in the debugger

The old perltodo notes that lvalue functions don't work in the debugger. This would be good to fix.

_ prototype character

Study the possibility of adding a new prototype character, _, meaning "this argument defaults to $_".

state variables

`my $foo if 0;` is deprecated, and should be replaced with `state $x = "initial value\n";` the syntax from Perl 6.

@INC source filter to Filter::Simple

The second return value from a sub in @INC can be a source filter. This isn't documented. It should be changed to use Filter::Simple, tested and documented.

regexp optimiser optional

The regexp optimiser is not optional. It should configurable to be, to allow its performance to be measured, and its bugs to be easily demonstrated.

UNITCHECK

Introduce a new special block, UNITCHECK, which is run at the end of a compilation unit (module, file, eval(STRING) block). This will correspond to the Perl 6 CHECK. Perl 5's CHECK cannot be changed or removed because the O.pm/B.pm backend framework depends on it.

optional optimizer

Make the peephole optimizer optional. Currently it performs two tasks as it walks the optree - genuine peephole optimisations, and necessary fixups of ops. It would be good to find an efficient way to switch out the optimisations whilst keeping the fixups.

You WANT *how* many

Currently contexts are void, scalar and list. split has a special mechanism in place to pass in the number of return values wanted. It would be useful to have a general mechanism for this, backwards compatible and little speed hit. This would allow proposals such as short circuiting sort to be implemented as a module on CPAN.

lexical aliases

Allow lexical aliases (maybe via the syntax `my \$alias = \$foo`.

entersub XS vs Perl

At the moment pp_entersub is huge, and has code to deal with entering both perl and XS subroutines. Subroutine implementations rarely change between perl and XS at run time, so investigate using 2 ops to enter subs (one for XS, one for perl) and swap between if a sub is redefined.

Self ties

self ties are currently illegal because they caused too many segfaults. Maybe the causes of these could be tracked down and self-ties on all types re- instated.

Optimize away @_

The old perltodo notes "Look at the "reification" code in `av.c`".

What hooks would assertions need?

Assertions are in the core, and work. However, assertions needed to be added as a core patch, rather than an XS module in ext, or a CPAN module, because the core has no hooks in the necessary places. It would be useful to investigate what hooks would need to be added to make it possible to provide the full assertion support from a CPAN module, so that we aren't constraining the imagination of future CPAN authors.

Big projects

Tasks that will get your name mentioned in the description of the "Highlights of 5.10"

make ithreads more robust

Generally make ithreads more robust. See also *iCOW*

This task is incremental - even a little bit of work on it will help, and will be greatly appreciated.

iCOW

Sarathy and Arthur have a proposal for an improved Copy On Write which specifically will be able to COW new ithreads. If this can be implemented it would be a good thing.

(?{...}) closures in regexps

Fix (or rewrite) the implementation of the `/(?{...})/` closures.

A re-entrant regexp engine

This will allow the use of a regex from inside (?{ }), (??{ }) and (?(?{ })|) constructs.