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?

Tasks that only need Perl knowledge

Remove duplication of test setup.

Schwern notes, that there's duplication of code - lots and lots of tests have some variation on the big block of `$Is_Foo` checks. We can safely put this into a file, change it to build an `%Is` hash and require it. Maybe just put it into *test.pl*. Throw in the handy tainting subroutines.

merge common code in installperl and installman

There are some common subroutines and a common `BEGIN` block in *installperl* and *installman*. These should probably be merged. It would also be good to check for duplication in all the utility scripts supplied in the source tarball. It might be good to move them all to a subdirectory, but this would require careful checking to find all places that call them, and change those correctly.

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.

merge checkpods and podchecker

pod/checkpods.PL (and `make check` in the *pod/* subdirectory) implements a very basic check for pod files, but the errors it discovers aren't found by podchecker. Add this check to podchecker, get rid of checkpods and have `make check` use podchecker.

perlmodlib.PL rewrite

Currently perlmodlib.PL needs to be run from a source directory where perl has been built, or some modules won't be found, and others will be skipped. Make it run from a clean perl source tree (so it's reproducible).

Parallel testing

(This probably impacts much more than the core: also the Test::Harness and TAP::* modules on CPAN.)

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 tests 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.

Improve the coverage of the core tests

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

test B

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

Deparse inlined constants

Code such as this

```
use constant PI => 4;
warn PI
```

will currently deparse as

```
use constant ('PI', 4);
warn 4;
```

because the tokenizer inlines the value of the constant subroutine `PI`. This allows various compile time optimisations, such as constant folding and dead code elimination. Where these haven't happened (such as the example above) it ought be possible to make B::Deparse work out the name of the original constant, because just enough information survives in the symbol table to do this. Specifically, the same scalar is used for the constant in the optree as is used for the constant subroutine, so by iterating over all symbol tables and generating a mapping of SV address to constant name, it would be possible to provide B::Deparse with this functionality.

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.

To make a minimal perl distribution, it's useful to look at *t/lib/commonsense.t*.

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.

embed.pl/makedef.pl

There is a script *embed.pl* that generates several header files to prefix all of Perl's symbols in a consistent way, to provide some semblance of namespace support in `C`. Functions are declared in *embed.fnc*, variables in *interpvar.h*. Quite a few of the functions and variables are conditionally declared there, using `#ifdef`. However, *embed.pl* doesn't understand the C macros, so the rules about which symbols are present when is duplicated in *makedef.pl*. Writing things twice is bad, m'kay. It would be good to teach `embed.pl` to understand the conditional compilation, and hence remove the duplication, and the mistakes it has caused.

use strict; and AutoLoad

Currently if you write

```
package Whack;
use AutoLoader 'AUTOLOAD';
use strict;
1;
__END__
sub bloop {
    print join (' ', No, strict, here), "!\n";
}
```

then `use strict;` isn't in force within the autoloaded subroutines. It would be more consistent (and less surprising) to arrange for all lexical pragmas in force at the __END__ block to be in force within each autoloaded subroutine.

There's a similar problem with SelfLoader.

Tasks that need a little sysadmin-type knowledge

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

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 built 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.
  ```

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.

Cross-compile support

Currently `Configure` understands `-Dusecrosscompile` option. This option arranges for building `miniperl` for TARGET machine, so this `miniperl` is assumed then to be copied to TARGET machine and used as a replacement of full `perl` executable.

This could be done little differently. Namely `miniperl` should be built for HOST and then full `perl` with extensions should be compiled for TARGET. This, however, might require extra trickery for %Config: we have one config first for HOST and then another for TARGET. Tools like MakeMaker will be mightily confused. Having around two different types of executables and libraries (HOST and TARGET) makes life interesting for Makefiles and shell (and Perl) scripts. There is $Config{run}, normally empty, which can be used as an execution wrapper. Also note that in some cross-compilation/execution environments the HOST and the TARGET do not see the same filesystem(s), the $Config{run} may need to do some file/directory copying back and forth.

roffitall

Make *pod/roffitall* be updated by *pod/buildtoc*.

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

Exterminate PL_na!

`PL_na` festers still in the darkest corners of various typemap files. It needs to be exterminated, replaced by a local variable of type `STRLEN`.

Modernize the order of directories in @INC

The way @INC is laid out by default, one cannot upgrade core (dual-life) modules without overwriting files. This causes problems for binary package builders. One possible proposal is laid out in this message: *http://www.xray.mpe.mpg.de/mailing-lists/perl5-porters/2002-04/msg02380.html*.

-Duse32bit*

Natively 64-bit systems need neither -Duse64bitint nor -Duse64bitall. On these systems, it might be the default compilation mode, and there is currently no guarantee that passing no use64bitall option to the Configure process will build a 32bit perl. Implementing -Duse32bit* options would be nice for perl 5.12.

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".

Profile Perl - am I hot or not?

The Perl source code is stable enough that it makes sense to profile it, identify and optimise the hotspots. It would be good to measure the performance of the Perl interpreter using free tools such as cachegrind, gprof, and dtrace, and work to reduce the bottlenecks they reveal.

As part of this, 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 as part of exercising your skills with coverage and profiling tools you 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*.

Allocate OPs from arenas

Currently all new OP structures are individually malloc()ed and free()d. All `malloc` implementations have space overheads, and are now as fast as custom allocates so it would both use less memory and less CPU to allocate the various OP structures from arenas. The SV arena code can probably be re-used for this.

Note that Configuring perl with `-Accflags=-DPL_OP_SLAB_ALLOC` will use Perl_Slab_alloc() to pack optrees into a contiguous block, which is probably superior to the use of OP arenas, esp. from a cache locality standpoint. See *Profile Perl - am I hot or not?*.

Improve win32/wince.c

Currently, numerous functions look virtually, if not completely, identical in both `win32/wince.c` and `win32/win32.c` files, which can't be good.

Use secure CRT functions when building with VC8 on Win32

Visual C++ 2005 (VC++ 8.x) deprecated a number of CRT functions on the basis that they were "unsafe" and introduced differently named secure versions of them as replacements, e.g. instead of writing

```
FILE* f = fopen(__FILE__, "r");
```

one should now write

```
FILE* f;
errno_t err = fopen_s(&f, __FILE__, "r");
```

Currently, the warnings about these deprecations have been disabled by adding -D_CRT_SECURE_NO_DEPRECATE to the CFLAGS. It would be nice to remove that warning suppressant and actually make use of the new secure CRT functions.

There is also a similar issue with POSIX CRT function names like fileno having been deprecated in favour of ISO C++ conformant names like _fileno. These warnings are also currently suppressed by adding -D_CRT_NONSTDC_NO_DEPRECATE. It might be nice to do as Microsoft suggest here too, although, unlike the secure functions issue, there is presumably little or no benefit in this case.

strcat(), strcpy(), strncat(), strncpy(), sprintf(), vsprintf()

Maybe create a utility that checks after each libperl.a creation that none of the above (nor sprintf(), vsprintf(), or *SHUDDER* gets()) ever creep back to libperl.a.

```
  nm libperl.a | ./miniperl -alne '$o = $F[0] if /:$/; print "$o $F[1]" if
$F[0] eq "U" && $F[1] =~ /^(?:strn?c(?:at|py)|v?sprintf|gets)$/'
```

Note, of course, that this will only tell whether **your** platform is using those naughty interfaces.

-D_FORTIFY_SOURCE=2, -fstack-protector

Recent glibcs support `-D_FORTIFY_SOURCE=2` and recent gcc (4.1 onwards?) supports `-fstack-protector`, both of which give protection against various kinds of buffer overflow problems. These should probably be used for compiling Perl whenever available, Configure and/or hints files should be adjusted to probe for the availability of these features and enable them as appropriate.

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.

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*.)

Most probably the right way to do this would be this: *Virtualize operating system access*.

Unicode in %ENV

Currently the %ENV entries are always byte strings. See *Virtualize operating system access*.

Unicode and glob()

Currently glob patterns and filenames returned from File::Glob::glob() are always byte strings. See *Virtualize operating system access*.

Unicode and lc/uc operators

Some built-in operators (`lc`, `uc`, etc.) behave differently, based on what the internal encoding of their argument is. That should not be the case. Maybe add a pragma to switch behaviour.

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.

Audit the code for destruction ordering assumptions

Change 25773 notes

```
/* Need to check SvMAGICAL, as during global destruction it may be that
   AvARYLEN(av) has been freed before av, and hence the SvANY() pointer
   is now part of the linked list of SV heads, rather than pointing to
   the original body.  */
/* FIXME - audit the code for other bugs like this one.  */
```

adding the `SvMAGICAL` check to

```
if (AvARYLEN(av) && SvMAGICAL(AvARYLEN(av))) {
    MAGIC *mg = mg_find (AvARYLEN(av), PERL_MAGIC_arylen);
```

Go through the core and look for similar assumptions that SVs have particular types, as all bets are off during global destruction.

Extend PerlIO and PerlIO::Scalar

PerlIO::Scalar doesn't know how to truncate(). Implementing this would require extending the PerlIO vtable.

Similarly the PerlIO vtable doesn't know about formats (write()), or about stat(), or chmod()/chown(), utime(), or flock().

(For PerlIO::Scalar it's hard to see what e.g. mode bits or ownership would mean.)

PerlIO doesn't do directories or symlinks, either: mkdir(), rmdir(), opendir(), closedir(), seekdir(), rewinddir(), glob(); symlink(), readlink().

See also *Virtualize operating system access*.

-C on the #! line

It should be possible to make -C work correctly if found on the #! line, given that all perl command line options are strict ASCII, and -C changes only the interpretation of non-ASCII characters, and not for the script file handle. To make it work needs some investigation of the ordering of function calls during startup, and (by implication) a bit of tweaking of that order.

Propagate const outwards from Perl_moreswitches()

Change 32057 changed the parameter and return value of `Perl_moreswitches()` from <char *> to <const char *>. It should now be possible to propagate const-correctness outwards to `S_parse_body()`, `Perl_moreswitches()` and `Perl_yylex()`.

Duplicate logic in S_method_common() and Perl_gv_fetchmethod_autoload()

A comment in `S_method_common` notes

```
 /* This code tries to figure out just what went wrong with
    gv_fetchmethod.  It therefore needs to duplicate a lot of
    the internals of that function.  We can't move it inside
    Perl_gv_fetchmethod_autoload(), however, since that would
    cause UNIVERSAL->can("NoSuchPackage::foo") to croak, and we
    don't want that.
 */
```

If `Perl_gv_fetchmethod_autoload` gets rewritten to take (more) flag bits, then it ought to be possible to move the logic from `S_method_common` to the "right" place. When making this change it would probably be good to also pass in at least the method name length, if not also pre-computed hash values when known. (I'm contemplating a plan to pre-compute hash values for common fixed strings such as `ISA` and pass them in to functions.)

Organize error messages

Perl's diagnostics (error messages, see *perldiag*) could use reorganizing and formalizing so that each error message has its stable-for-all-eternity unique id, categorized by severity, type, and subsystem. (The error messages would be listed in a datafile outside of the Perl source code, and the source code would only refer to the messages by the id.) This clean-up and regularizing should apply for all croak() messages.

This would enable all sorts of things: easier translation/localization of the messages (though please do keep in mind the caveats of *Locale::Maketext* about too straightforward approaches to translation), filtering by severity, and instead of grepping for a particular error message one could look for a stable error id. (Of course, changing the error messages by default would break all the existing software depending on some particular error message...)

This kind of functionality is known as *message catalogs*. Look for inspiration for example in the catgets() system, possibly even use it if available-- but **only** if available, all platforms will **not** have catgets().

For the really pure at heart, consider extending this item to cover also the warning messages (see *perllexwarn*, `warnings.pl`).

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.

UTF-8 revamp

The handling of Unicode is unclean in many places. For example, the regexp engine matches in Unicode semantics whenever the string or the pattern is flagged as UTF-8, but that should not be dependent on an internal storage detail of the string. Likewise, case folding behaviour is dependent on the UTF8 internal flag being on or off.

Properly Unicode safe tokeniser and pads.

The tokeniser isn't actually very UTF-8 clean. `use utf8;` is a hack - variable names are stored in stashes as raw bytes, without the utf-8 flag set. The pad API only takes a `char *` pointer, so that's all bytes too. The tokeniser ignores the UTF-8-ness of `PL_rsfp`, or any SVs returned from source filters. All this could be fixed.

state variable initialization in list context

Currently this is illegal:

```
    state ($a, $b) = foo();
```

In Perl 6, `state ($a) = foo();` and `(state $a) = foo();` have different semantics, which is tricky to implement in Perl 5 as currently they produce the same opcode trees. The Perl 6 design is firm, so it would be good to implement the necessary code in Perl 5. There are comments in `Perl_newASSIGNOP()` that show the code paths taken by various assignment constructions involving state variables.

Implement $value ~~ 0 .. $range

It would be nice to extend the syntax of the ~~ operator to also understand numeric (and maybe alphanumeric) ranges.

A does() built-in

Like ref(), only useful. It would call the `DOES` method on objects; it would also tell whether something can be dereferenced as an array/hash/etc., or used as a regexp, etc.
http://www.xray.mpe.mpg.de/mailing-lists/perl5-porters/2007-03/msg00481.html

Tied filehandles and write() don't mix

There is no method on tied filehandles to allow them to be called back by formats.

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.

Optimize away empty destructors

Defining an empty DESTROY method might be useful (notably in AUTOLOAD-enabled classes), but it's still a bit expensive to call. That could probably be optimized.

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.

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.

delete &function

Allow to delete functions. One can already undef them, but they're still in the stash.

/w regex modifier

That flag would enable to match whole words, and also to interpolate arrays as alternations. With it, `/P/w` would be roughly equivalent to:

```
do { local $"='|'; /\b(?:P)\b/ }
```

See *http://www.xray.mpe.mpg.de/mailing-lists/perl5-porters/2007-01/msg00400.html* for the discussion.

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 reinstated.

Optimize away @_

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

The yada yada yada operators

Perl 6's Synopsis 3 says:

The ... operator is the "yada, yada, yada" list operator, which is used as the body in function prototypes. It complains bitterly (by calling fail) if it is ever executed. Variant ??? calls warn, and !!! calls die.

Those would be nice to add to Perl 5. That could be done without new ops.

Virtualize operating system access

Implement a set of "vtables" that virtualizes operating system access (open(), mkdir(), unlink(), readdir(), getenv(), etc.) At the very least these interfaces should take SVs as "name" arguments instead of bare char pointers; probably the most flexible and extensible way would be for the Perl-facing interfaces to accept HVs. The system needs to be per-operating-system and per-file-system hookable/filterable, preferably both from XS and Perl level (*"Files and Filesystems" in perlport* is good reading at this point, in fact, all of *perlport* is.)

This has actually already been implemented (but only for Win32), take a look at *iperlsys.h* and *win32/perlhost.h*. While all Win32 variants go through a set of "vtables" for operating system access, non-Win32 systems currently go straight for the POSIX/UNIX-style system/library call. Similar system as for Win32 should be implemented for all platforms. The existing Win32 implementation probably does not need to survive alongside this proposed new implementation, the approaches could be merged.

What would this give us? One often-asked-for feature this would enable is using Unicode for filenames, and other "names" like %ENV, usernames, hostnames, and so forth. (See *"When Unicode Does Not Happen" in perlunicode*.)

But this kind of virtualization would also allow for things like virtual filesystems, virtual networks, and "sandboxes" (though as long as dynamic loading of random object code is allowed, not very safe sandboxes since external code of course know not of Perl's vtables). An example of a smaller "sandbox" is that this feature can be used to implement per-thread working directories: Win32 already does this.

See also *Extend PerlIO and PerlIO::Scalar*.

Investigate PADTMP hash pessimisation

The peephole optimier converts constants used for hash key lookups to shared hash key scalars. Under ithreads, something is undoing this work. See See http://www.xray.mpe.mpg.de/mailing-lists/perl5-porters/2007-09/msg00793.html

Big projects

Tasks that will get your name mentioned in the description of the "Highlights of 5.12"

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.

One bit would be to write the missing code in sv.c:Perl_dirp_dup.

Fix Perl_sv_dup, et al so that threads can return objects.

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.

Add class set operations to regexp engine

Apparently these are quite useful. Anyway, Jeffery Friedl wants them.

demerphq has this on his todo list, but right at the bottom.