NAME

perldelta - what is new for perl v5.22.0

DESCRIPTION

This document describes differences between the 5.20.0 release and the 5.22.0 release.

If you are upgrading from an earlier release such as 5.18.0, first read *perl5200delta*, which describes differences between 5.18.0 and 5.20.0.

Core Enhancements

New bitwise operators

A new experimental facility has been added that makes the four standard bitwise operators (`& | ^ ~`) treat their operands consistently as numbers, and introduces four new dotted operators (`&. |. ^. ~.`) that treat their operands consistently as strings. The same applies to the assignment variants (`&= |= ^= &.= |.= ^.=`).

To use this, enable the "bitwise" feature and disable the "experimental::bitwise" warnings category. See *"Bitwise String Operators" in perlop* for details. *[perl #123466]*.

New double-diamond operator

`<<>>` is like `<>` but uses three-argument `open` to open each file in `@ARGV`. This means that each element of `@ARGV` will be treated as an actual file name, and `"|foo"` won't be treated as a pipe open.

New \b boundaries in regular expressions

qr/\b{gcb}/

`gcb` stands for Grapheme Cluster Boundary. It is a Unicode property that finds the boundary between sequences of characters that look like a single character to a native speaker of a language. Perl has long had the ability to deal with these through the `\X` regular escape sequence. Now, there is an alternative way of handling these. See *"\b{}, \b, \B{}, \B" in perlrebackslash* for details.

qr/\b{wb}/

`wb` stands for Word Boundary. It is a Unicode property that finds the boundary between words. This is similar to the plain `\b` (without braces) but is more suitable for natural language processing. It knows, for example, that apostrophes can occur in the middle of words. See *"\b{}, \b, \B{}, \B" in perlrebackslash* for details.

qr/\b{sb}/

`sb` stands for Sentence Boundary. It is a Unicode property to aid in parsing natural language sentences. See *"\b{}, \b, \B{}, \B" in perlrebackslash* for details.

Non-Capturing Regular Expression Flag

Regular expressions now support a `/n` flag that disables capturing and filling in `$1`, `$2`, etc inside of groups:

```
"hello" =~ /(hi|hello)/n; # $1 is not set
```

This is equivalent to putting `?:` at the beginning of every capturing group.

See *"n" in perlre* for more information.

use re 'strict'

This applies stricter syntax rules to regular expression patterns compiled within its scope. This will hopefully alert you to typos and other unintentional behavior that backwards-compatibility issues prevent us from reporting in normal regular expression compilations. Because the behavior of this is subject to change in future Perl releases as we gain experience, using this pragma will raise a warning of category `experimental::re_strict`. See *'strict' in re*.

Unicode 7.0 (with correction) is now supported

For details on what is in this release, see *http://www.unicode.org/versions/Unicode7.0.0/*. The version of Unicode 7.0 that comes with Perl includes a correction dealing with glyph shaping in Arabic (see *http://www.unicode.org/errata/#current_errata*).

use locale can restrict which locale categories are affected

It is now possible to pass a parameter to `use locale` to specify a subset of locale categories to be locale-aware, with the remaining ones unaffected. See *"The "use locale" pragma" in perllocale* for details.

Perl now supports POSIX 2008 locale currency additions

On platforms that are able to handle POSIX.1-2008, the hash returned by *`POSIX::localeconv()`* includes the international currency fields added by that version of the POSIX standard. These are `int_n_cs_precedes`, `int_n_sep_by_space`, `int_n_sign_posn`, `int_p_cs_precedes`, `int_p_sep_by_space`, and `int_p_sign_posn`.

Better heuristics on older platforms for determining locale UTF-8ness

On platforms that implement neither the C99 standard nor the POSIX 2001 standard, determining if the current locale is UTF-8 or not depends on heuristics. These are improved in this release.

Aliasing via reference

Variables and subroutines can now be aliased by assigning to a reference:

```
\$c = \$d;
\&x = \&y;
```

Aliasing can also be accomplished by using a backslash before a `foreach` iterator variable; this is perhaps the most useful idiom this feature provides:

```
foreach \%hash (@array_of_hash_refs) { ... }
```

This feature is experimental and must be enabled via `use feature 'refaliasing'`. It will warn unless the `experimental::refaliasing` warnings category is disabled.

See *"Assigning to References" in perlref*

prototype with no arguments

`prototype()` with no arguments now infers `$_`. *[perl #123514]*.

New :const subroutine attribute

The `const` attribute can be applied to an anonymous subroutine. It causes the new sub to be executed immediately whenever one is created (*i.e.* when the `sub` expression is evaluated). Its value is captured and used to create a new constant subroutine that is returned. This feature is experimental. See *"Constant Functions" in perlsub*.

fileno now works on directory handles

When the relevant support is available in the operating system, the `fileno` builtin now works on directory handles, yielding the underlying file descriptor in the same way as for filehandles. On operating systems without such support, `fileno` on a directory handle continues to return the undefined value, as before, but also sets `$!` to indicate that the operation is not supported.

Currently, this uses either a `dd_fd` member in the OS `DIR` structure, or a `dirfd(3)` function as specified by POSIX.1-2008.

List form of pipe open implemented for Win32

The list form of pipe:

```
open my $fh, "-|", "program", @arguments;
```

is now implemented on Win32. It has the same limitations as `system LIST` on Win32, since the Win32 API doesn't accept program arguments as a list.

Assignment to list repetition

`(...) x ...` can now be used within a list that is assigned to, as long as the left-hand side is a valid lvalue. This allows `(undef,undef,$foo) = that_function()` to be written as `((undef)x2, $foo) = that_function()`.

Infinity and NaN (not-a-number) handling improved

Floating point values are able to hold the special values infinity, negative infinity, and NaN (not-a-number). Now we more robustly recognize and propagate the value in computations, and on output normalize them to the strings `Inf`, `-Inf`, and `NaN`.

See also the *POSIX* enhancements.

Floating point parsing has been improved

Parsing and printing of floating point values has been improved.

As a completely new feature, hexadecimal floating point literals (like `0x1.23p-4`) are now supported, and they can be output with `printf "%a"`. See *"Scalar value constructors" in perldata* for more details.

Packing infinity or not-a-number into a character is now fatal

Before, when trying to pack infinity or not-a-number into a (signed) character, Perl would warn, and assumed you tried to pack `0xFF`; if you gave it as an argument to `chr`, `U+FFFD` was returned.

But now, all such actions (`pack`, `chr`, and `print '%c'`) result in a fatal error.

Experimental C Backtrace API

Perl now supports (via a C level API) retrieving the C level backtrace (similar to what symbolic debuggers like gdb do).

The backtrace returns the stack trace of the C call frames, with the symbol names (function names), the object names (like "perl"), and if it can, also the source code locations (file:line).

The supported platforms are Linux and OS X (some *BSD might work at least partly, but they have not yet been tested).

The feature needs to be enabled with `Configure -Dusecbacktrace`.

See *"C backtrace" in perlhacktips* for more information.

Security

Perl is now compiled with -fstack-protector-strong if available

Perl has been compiled with the anti-stack-smashing option `-fstack-protector` since 5.10.1. Now Perl uses the newer variant called `-fstack-protector-strong`, if available.

The Safe module could allow outside packages to be replaced

Critical bugfix: outside packages could be replaced. *Safe* has been patched to 2.38 to address this.

Perl is now always compiled with -D_FORTIFY_SOURCE=2 if available

The 'code hardening' option called `_FORTIFY_SOURCE`, available in gcc 4.*, is now always used for compiling Perl, if available.

Note that this isn't necessarily a huge step since in many platforms the step had already been taken several years ago: many Linux distributions (like Fedora) have been using this option for Perl, and OS X has enforced the same for many years.

Incompatible Changes

Subroutine signatures moved before attributes

The experimental sub signatures feature, as introduced in 5.20, parsed signatures after attributes. In this release, following feedback from users of the experimental feature, the positioning has been moved such that signatures occur after the subroutine name (if any) and before the attribute list (if any).

& and \& prototypes accepts only subs

The & prototype character now accepts only anonymous subs (`sub {...}`), things beginning with `\&`, or an explicit `undef`. Formerly it erroneously also allowed references to arrays, hashes, and lists. *[perl #4539]*. *[perl #123062]*. *[perl #123062]*.

In addition, the `\&` prototype was allowing subroutine calls, whereas now it only allows subroutines: `&foo` is still permitted as an argument, while `&foo()` and `foo()` no longer are. *[perl #77860]*.

use encoding is now lexical

The *encoding* pragma's effect is now limited to lexical scope. This pragma is deprecated, but in the meantime, it could adversely affect unrelated modules that are included in the same program; this change fixes that.

List slices returning empty lists

List slices now return an empty list only if the original list was empty (or if there are no indices). Formerly, a list slice would return an empty list if all indices fell outside the original list; now it returns a list of `undef` values in that case. *[perl #114498]*.

\N{} with a sequence of multiple spaces is now a fatal error

E.g. `\N{TOO  MANY SPACES}` or `\N{TRAILING SPACE }`. This has been deprecated since v5.18.

use UNIVERSAL '...' is now a fatal error

Importing functions from `UNIVERSAL` has been deprecated since v5.12, and is now a fatal error. `use UNIVERSAL` without any arguments is still allowed.

In double-quotish \cX, X must now be a printable ASCII character

In prior releases, failure to do this raised a deprecation warning.

Splitting the tokens (? and (* in regular expressions is now a fatal compilation error.

These had been deprecated since v5.18.

qr/foo/x now ignores all Unicode pattern white space

The `/x` regular expression modifier allows the pattern to contain white space and comments (both of which are ignored) for improved readability. Until now, not all the white space characters that Unicode designates for this purpose were handled. The additional ones now recognized are:

```
U+0085 NEXT LINE
U+200E LEFT-TO-RIGHT MARK
U+200F RIGHT-TO-LEFT MARK
U+2028 LINE SEPARATOR
U+2029 PARAGRAPH SEPARATOR
```

The use of these characters with `/x` outside bracketed character classes and when not preceded by a backslash has raised a deprecation warning since v5.18. Now they will be ignored.

Comment lines within (?[ ]) are now ended only by a \n

`(?[ ])` is an experimental feature, introduced in v5.18. It operates as if `/x` is always enabled. But there was a difference: comment lines (following a # character) were terminated by anything matching `\R` which includes all vertical whitespace, such as form feeds. For consistency, this is now changed to match what terminates comment lines outside `(?[ ])`, namely a `\n` (even if escaped), which is the same as what terminates a heredoc string and formats.

(?[...]) operators now follow standard Perl precedence

This experimental feature allows set operations in regular expression patterns. Prior to this, the intersection operator had the same precedence as the other binary operators. Now it has higher precedence. This could lead to different outcomes than existing code expects (though the documentation has always noted that this change might happen, recommending fully parenthesizing the expressions). See *"Extended Bracketed Character Classes" in perlrecharclass*.

Omitting % and @ on hash and array names is no longer permitted

Really old Perl let you omit the `@` on array names and the `%` on hash names in some spots. This has issued a deprecation warning since Perl 5.000, and is no longer permitted.

"$!" text is now in English outside the scope of use locale

Previously, the text, unlike almost everything else, always came out based on the current underlying locale of the program. (Also affected on some systems is `"$^E"`.) For programs that are unprepared to handle locale differences, this can cause garbage text to be displayed. It's better to display text that is translatable via some tool than garbage text which is much harder to figure out.

"$!" text will be returned in UTF-8 when appropriate

The stringification of `$!` and `$^E` will have the UTF-8 flag set when the text is actually non-ASCII UTF-8. This will enable programs that are set up to be locale-aware to properly output messages in the user's native language. Code that needs to continue the 5.20 and earlier behavior can do the stringification within the scopes of both `use bytes` and `use locale ":messages"`. Within these two scopes, no other Perl operations will be affected by locale; only `$!` and `$^E` stringification. The `bytes` pragma causes the UTF-8 flag to not be set, just as in previous Perl releases. This resolves *[perl #112208]*.

Support for ?PATTERN? without explicit operator has been removed

The `m?PATTERN?` construct, which allows matching a regex only once, previously had an alternative form that was written directly with a question mark delimiter, omitting the explicit `m` operator. This usage has produced a deprecation warning since 5.14.0. It is now a syntax error, so that the question mark can be available for use in new operators.

defined(@array) and defined(%hash) are now fatal errors

These have been deprecated since v5.6.1 and have raised deprecation warnings since v5.16.

Using a hash or an array as a reference are now fatal errors

For example, `%foo->{"bar"}` now causes a fatal compilation error. These have been deprecated since before v5.8, and have raised deprecation warnings since then.

Changes to the * prototype

The * character in a subroutine's prototype used to allow barewords to take precedence over most, but not all, subroutine names. It was never consistent and exhibited buggy behavior.

Now it has been changed, so subroutines always take precedence over barewords, which brings it into conformity with similarly prototyped built-in functions:

```
sub splat(*) { ... }
sub foo { ... }
splat(foo); # now always splat(foo())
```

```
splat(bar); # still splat('bar') as before
close(foo); # close(foo())
close(bar); # close('bar')
```

Deprecations

Setting ${^ENCODING} to anything but undef

This variable allows Perl scripts to be written in an encoding other than ASCII or UTF-8. However, it affects all modules globally, leading to wrong answers and segmentation faults. New scripts should be written in UTF-8; old scripts should be converted to UTF-8, which is easily done with the *piconv* utility.

Use of non-graphic characters in single-character variable names

The syntax for single-character variable names is more lenient than for longer variable names, allowing the one-character name to be a punctuation character or even invisible (a non-graphic). Perl v5.20 deprecated the ASCII-range controls as such a name. Now, all non-graphic characters that formerly were allowed are deprecated. The practical effect of this occurs only when not under `use utf8`, and affects just the C1 controls (code points 0x80 through 0xFF), NO-BREAK SPACE, and SOFT HYPHEN.

Inlining of sub () { $var } with observable side-effects

In many cases Perl makes `sub () { $var }` into an inlinable constant subroutine, capturing the value of `$var` at the time the `sub` expression is evaluated. This can break the closure behavior in those cases where `$var` is subsequently modified, since the subroutine won't return the changed value. (Note that this all only applies to anonymous subroutines with an empty prototype (`sub ()`).)

This usage is now deprecated in those cases where the variable could be modified elsewhere. Perl detects those cases and emits a deprecation warning. Such code will likely change in the future and stop producing a constant.

If your variable is only modified in the place where it is declared, then Perl will continue to make the sub inlinable with no warnings.

```
sub make_constant {
    my $var = shift;
    return sub () { $var }; # fine
}

sub make_constant_deprecated {
    my $var;
    $var = shift;
    return sub () { $var }; # deprecated
}

sub make_constant_deprecated2 {
    my $var = shift;
    log_that_value($var); # could modify $var
    return sub () { $var }; # deprecated
}
```

In the second example above, detecting that `$var` is assigned to only once is too hard to detect. That it happens in a spot other than the `my` declaration is enough for Perl to find it suspicious.

This deprecation warning happens only for a simple variable for the body of the sub. (A `BEGIN` block or `use` statement inside the sub is ignored, because it does not become part of the sub's body.) For more complex cases, such as `sub () { do_something() if 0; $var }` the behavior has changed such that inlining does not happen if the variable is modifiable elsewhere. Such cases

should be rare.

Use of multiple /x regexp modifiers

It is now deprecated to say something like any of the following:

```
qr/foo/xx;
/(?xax:foo)/;
use re qw(/amxx);
```

That is, now `x` should only occur once in any string of contiguous regular expression pattern modifiers. We do not believe there are any occurrences of this in all of CPAN. This is in preparation for a future Perl release having `/xx` permit white-space for readability in bracketed character classes (those enclosed in square brackets: `[...]`).

Using a NO-BREAK space in a character alias for \N{...} is now deprecated

This non-graphic character is essentially indistinguishable from a regular space, and so should not be allowed. See *"CUSTOM ALIASES" in charnames*.

A literal "{" should now be escaped in a pattern

If you want a literal left curly bracket (also called a left brace) in a regular expression pattern, you should now escape it by either preceding it with a backslash (`"\{"`) or enclosing it within square brackets `"[{]"`, or by using `\Q`; otherwise a deprecation warning will be raised. This was first announced as forthcoming in the v5.16 release; it will allow future extensions to the language to happen.

Making all warnings fatal is discouraged

The documentation for *fatal warnings* notes that `use warnings FATAL => 'all'` is discouraged, and provides stronger language about the risks of fatal warnings in general.

Performance Enhancements

- If a method or class name is known at compile time, a hash is precomputed to speed up run-time method lookup. Also, compound method names like `SUPER::new` are parsed at compile time, to save having to parse them at run time.
- Array and hash lookups (especially nested ones) that use only constants or simple variables as keys, are now considerably faster. See *Internal Changes* for more details.
- `(...)x1`, `("constant")x0` and `($scalar)x0` are now optimised in list context. If the right-hand argument is a constant 1, the repetition operator disappears. If the right-hand argument is a constant 0, the whole expression is optimised to the empty list, so long as the left-hand argument is a simple scalar or constant. (That is, `(foo())x0` is not subject to this optimisation.)
- `substr` assignment is now optimised into 4-argument `substr` at the end of a subroutine (or as the argument to `return`). Previously, this optimisation only happened in void context.
- In `"\L..."`, `"\Q..."`, etc., the extra "stringify" op is now optimised away, making these just as fast as `lcfirst`, `quotemeta`, etc.
- Assignment to an empty list is now sometimes faster. In particular, it never calls `FETCH` on tied arguments on the right-hand side, whereas it used to sometimes.
- There is a performance improvement of up to 20% when `length` is applied to a non-magical, non-tied string, and either `use bytes` is in scope or the string doesn't use UTF-8 internally.
- On most perl builds with 64-bit integers, memory usage for non-magical, non-tied scalars containing only a floating point value has been reduced by between 8 and 32 bytes, depending on OS.

- In `@array = split`, the assignment can be optimized away, so that `split` writes directly to the array. This optimisation was happening only for package arrays other than `@_`, and only sometimes. Now this optimisation happens almost all the time.
- `join` is now subject to constant folding. So for example `join "-", "a", "b"` is converted at compile-time to `"a-b"`. Moreover, `join` with a scalar or constant for the separator and a single-item list to join is simplified to a stringification, and the separator doesn't even get evaluated.
- `qq(@array)` is implemented using two ops: a stringify op and a join op. If the `qq` contains nothing but a single array, the stringification is optimized away.
- `our $var` and `our($s,@a,%h)` in void context are no longer evaluated at run time. Even a whole sequence of `our $foo;` statements will simply be skipped over. The same applies to `state` variables.
- Many internal functions have been refactored to improve performance and reduce their memory footprints. *[perl #121436] [perl #121906] [perl #121969]*
- `-T` and `-B` filetests will return sooner when an empty file is detected. *[perl #121489]*
- Hash lookups where the key is a constant are faster.
- Subroutines with an empty prototype and a body containing just `undef` are now eligible for inlining. *[perl #122728]*
- Subroutines in packages no longer need to be stored in typeglobs: declaring a subroutine will now put a simple sub reference directly in the stash if possible, saving memory. The typeglob still notionally exists, so accessing it will cause the stash entry to be upgraded to a typeglob (*i.e.* this is just an internal implementation detail). This optimization does not currently apply to XSUBs or exported subroutines, and method calls will undo it, since they cache things in typeglobs. *[perl #120441]*
- The functions `utf8::native_to_unicode()` and `utf8::unicode_to_native()` (see *utf8*) are now optimized out on ASCII platforms. There is now not even a minimal performance hit in writing code portable between ASCII and EBCDIC platforms.
- Win32 Perl uses 8 KB less of per-process memory than before for every perl process, because some data is now memory mapped from disk and shared between processes from the same perl binary.

Modules and Pragmata

Updated Modules and Pragmata

Many of the libraries distributed with perl have been upgraded since v5.20.0. For a complete list of changes, run:

```
corelist --diff 5.20.0 5.22.0
```

You can substitute your favorite version in place of 5.20.0, too.

Some notable changes include:

- *Archive::Tar* has been upgraded to version 2.04.

  Tests can now be run in parallel.

- *attributes* has been upgraded to version 0.27.

  The usage of `memEQ`s in the XS has been corrected. *[perl #122701]*

  Avoid reading beyond the end of a buffer. [perl #122629]

- *B* has been upgraded to version 1.58.

  It provides a new `B::safename` function, based on the existing `B::GV->SAFENAME`, that converts `\cOPEN` to `^OPEN`.

  Nulled COPs are now of class `B::COP`, rather than `B::OP`.

  `B::REGEXP` objects now provide a `qr_anoncv` method for accessing the implicit CV associated with `qr//` things containing code blocks, and a `compflags` method that returns the pertinent flags originating from the `qr//blahblah` op.

  `B::PMOP` now provides a `pmregexp` method returning a `B::REGEXP` object. Two new classes, `B::PADNAME` and `B::PADNAMELIST`, have been introduced.

  A bug where, after an ithread creation or psuedofork, special/immortal SVs in the child ithread/psuedoprocess did not have the correct class of `B::SPECIAL`, has been fixed. The `id` and `outid` PADLIST methods have been added.

- *B::Concise* has been upgraded to version 0.996.

  Null ops that are part of the execution chain are now given sequence numbers.

  Private flags for nulled ops are now dumped with mnemonics as they would be for the non-nulled counterparts.

- *B::Deparse* has been upgraded to version 1.35.

  It now deparses `+sub : attr { ... }` correctly at the start of a statement. Without the initial `+`, `sub` would be a statement label.

  `BEGIN` blocks are now emitted in the right place most of the time, but the change unfortunately introduced a regression, in that `BEGIN` blocks occurring just before the end of the enclosing block may appear below it instead.

  `B::Deparse` no longer puts erroneous `local` here and there, such as for `LIST = tr/a//d`. [perl #119815]

  Adjacent `use` statements are no longer accidentally nested if one contains a `do` block. [perl #115066]

  Parenthesised arrays in lists passed to `\` are now correctly deparsed with parentheses (*e.g.*, `\(@a, (@b), @c)` now retains the parentheses around @b), thus preserving the flattening behavior of referenced parenthesised arrays. Formerly, it only worked for one array: `\(@a)`.

  `local our` is now deparsed correctly, with the `our` included.

  `for($foo; !$bar; $baz) {...}` was deparsed without the `!` (or `not`). This has been fixed.

  Core keywords that conflict with lexical subroutines are now deparsed with the `CORE::` prefix.

  `foreach state $x (...) {...}` now deparses correctly with `state` and not `my`.

  `our @array = split(...)` now deparses correctly with `our` in those cases where the assignment is optimized away.

  It now deparses `our(`*`LIST`*`)` and typed lexical (`my Dog $spot`) correctly.

  Deparse `$#_` as that instead of as `$#{_}`. *[perl #123947]*

  BEGIN blocks at the end of the enclosing scope are now deparsed in the right place. [perl #77452]

  BEGIN blocks were sometimes deparsed as __ANON__, but are now always called BEGIN.

  Lexical subroutines are now fully deparsed. [perl #116553]

  `Anything =~ y///r` with `/r` no longer omits the left-hand operand.

  The op trees that make up regexp code blocks are now deparsed for real. Formerly, the original string that made up the regular expression was used. That caused problems with `qr/(?{<<heredoc})/` and multiline code blocks, which were deparsed incorrectly. [perl #123217] [perl #115256]

`$;` at the end of a statement no longer loses its semicolon. [perl #123357]

Some cases of subroutine declarations stored in the stash in shorthand form were being omitted.

Non-ASCII characters are now consistently escaped in strings, instead of some of the time. (There are still outstanding problems with regular expressions and identifiers that have not been fixed.)

When prototype sub calls are deparsed with `&` (*e.g.*, under the **-P** option), `scalar` is now added where appropriate, to force the scalar context implied by the prototype.

`require(foo())`, `do(foo())`, `goto(foo())` and similar constructs with loop controls are now deparsed correctly. The outer parentheses are not optional.

Whitespace is no longer escaped in regular expressions, because it was getting erroneously escaped within `(?x:...)` sections.

`sub foo { foo() }` is now deparsed with those mandatory parentheses.

`/@array/` is now deparsed as a regular expression, and not just `@array`.

`/@{-}/`, `/@{+}/` and `$#{1}` are now deparsed with the braces, which are mandatory in these cases.

In deparsing feature bundles, `B::Deparse` was emitting `no feature;` first instead of `no feature ':all';`. This has been fixed.

`chdir FH` is now deparsed without quotation marks.

`\my @a` is now deparsed without parentheses. (Parenthese would flatten the array.)

`system` and `exec` followed by a block are now deparsed correctly. Formerly there was an erroneous `do` before the block.

`use constant QR => qr/.../flags` followed by `"" =~ QR` is no longer without the flags.

Deparsing `BEGIN { undef &foo }` with the **-w** switch enabled started to emit 'uninitialized' warnings in Perl 5.14. This has been fixed.

Deparsing calls to subs with a `(;+)` prototype resulted in an infinite loop. The `(;$)` `(_)` and `(;_)` prototypes were given the wrong precedence, causing `foo($a<$b)` to be deparsed without the parentheses.

Deparse now provides a defined state sub in inner subs.

- *B::Op_private* has been added.

  B::Op_private provides detailed information about the flags used in the `op_private` field of perl opcodes.

- *bigint*, *bignum*, *bigrat* have been upgraded to version 0.39.

  Document in CAVEATS that using strings as numbers won't always invoke the big number overloading, and how to invoke it. [rt.perl.org #123064]

- *Carp* has been upgraded to version 1.36.

  `Carp::Heavy` now ignores version mismatches with Carp if Carp is newer than 1.12, since `Carp::Heavy`'s guts were merged into Carp at that point. *[perl #121574]*

  Carp now handles non-ASCII platforms better.

  Off-by-one error fix for Perl < 5.14.

- *constant* has been upgraded to version 1.33.

  It now accepts fully-qualified constant names, allowing constants to be defined in packages other than the caller.

- *CPAN* has been upgraded to version 2.11.

Add support for `Cwd::getdcwd()` and introduce workaround for a misbehavior seen on Strawberry Perl 5.20.1.

Fix `chdir()` after building dependencies bug.

Introduce experimental support for plugins/hooks.

Integrate the `App::Cpan` sources.

Do not check recursion on optional dependencies.

Sanity check *META.yml* to contain a hash. *[cpan #95271]*

- *CPAN::Meta::Requirements* has been upgraded to version 2.132.

  Works around limitations in `version::vpp` detecting v-string magic and adds support for forthcoming *ExtUtils::MakeMaker* bootstrap *version.pm* for Perls older than 5.10.0.

- *Data::Dumper* has been upgraded to version 2.158.

  Fixes CVE-2014-4330 by adding a configuration variable/option to limit recursion when dumping deep data structures.

  Changes to resolve Coverity issues. XS dumps incorrectly stored the name of code references stored in a GLOB. *[perl #122070]*

- *DynaLoader* has been upgraded to version 1.32.

  Remove `dl_nonlazy` global if unused in Dynaloader. [perl #122926]

- *Encode* has been upgraded to version 2.72.

  `piconv` now has better error handling when the encoding name is nonexistent, and a build breakage when upgrading *Encode* in perl-5.8.2 and earlier has been fixed.

  Building in C++ mode on Windows now works.

- *Errno* has been upgraded to version 1.23.

  Add `-P` to the preprocessor command-line on GCC 5. GCC added extra line directives, breaking parsing of error code definitions. [rt.perl.org #123784]

- *experimental* has been upgraded to version 0.013.

  Hardcodes features for Perls older than 5.15.7.

- *ExtUtils::CBuilder* has been upgraded to version 0.280221.

  Fixes a regression on Android. *[perl #122675]*

- *ExtUtils::Manifest* has been upgraded to version 1.70.

  Fixes a bug with `maniread()`'s handling of quoted filenames and improves `manifind()` to follow symlinks. *[perl #122415]*

- *ExtUtils::ParseXS* has been upgraded to version 3.28.

  Only declare `file` unused if we actually define it. Improve generated `RETVAL` code generation to avoid repeated references to `ST(0)`. [perl #123278] Broaden and document the `/OBJ$/` to `/REF$/` typemap optimization for the `DESTROY` method. [perl #123418]

- *Fcntl* has been upgraded to version 1.13.

  Add support for the Linux pipe buffer size `fcntl()` commands.

- *File::Find* has been upgraded to version 1.29.

  `find()` and `finddepth()` will now warn if passed inappropriate or misspelled options.

- *File::Glob* has been upgraded to version 1.24.

  Avoid `SvIV()` expanding to call `get_sv()` three times in a few places. [perl #123606]

- *HTTP::Tiny* has been upgraded to version 0.054.

  `keep_alive` is now fork-safe and thread-safe.

- *IO* has been upgraded to version 1.35.

  The XS implementation has been fixed for the sake of older Perls.

- *IO::Socket* has been upgraded to version 1.38.

  Document the limitations of the `connected()` method. [perl #123096]

- *IO::Socket::IP* has been upgraded to version 0.37.

  A better fix for subclassing `connect()`. *[cpan #95983] [cpan #97050]*

  Implements Timeout for `connect()`. *[cpan #92075]*

- The libnet collection of modules has been upgraded to version 3.05.

  Support for IPv6 and SSL to `Net::FTP`, `Net::NNTP`, `Net::POP3` and `Net::SMTP`. Improvements in `Net::SMTP` authentication.

- *Locale::Codes* has been upgraded to version 3.34.

  Fixed a bug in the scripts used to extract data from spreadsheets that prevented the SHP currency code from being found. *[cpan #94229]*

  New codes have been added.

- *Math::BigInt* has been upgraded to version 1.9997.

  Synchronize POD changes from the CPAN release. `Math::BigFloat->blog(x)` would sometimes return `blog(2*x)` when the accuracy was greater than 70 digits. The result of `Math::BigFloat->bdiv()` in list context now satisfies `x = quotient * divisor + remainder`.

  Correct handling of subclasses. *[cpan #96254] [cpan #96329]*

- *Module::Metadata* has been upgraded to version 1.000026.

  Support installations on older perls with an *ExtUtils::MakeMaker* earlier than 6.63_03

- *overload* has been upgraded to version 1.26.

  A redundant `ref $sub` check has been removed.

- The PathTools module collection has been upgraded to version 3.56.

  A warning from the **gcc** compiler is now avoided when building the XS.

  Don't turn leading `//` into `/` on Cygwin. [perl #122635]

- *perl5db.pl* has been upgraded to version 1.49.

  The debugger would cause an assertion failure. *[perl #124127]*

  `fork()` in the debugger under `tmux` will now create a new window for the forked process. *[perl #121333]*

  The debugger now saves the current working directory on startup and restores it when you restart your program with `R` or `rerun`. *[perl #121509]*

- *PerlIO::scalar* has been upgraded to version 0.22.

  Reading from a position well past the end of the scalar now correctly returns end of file. [perl #123443]

  Seeking to a negative position still fails, but no longer leaves the file position set to a negation location.

  `eof()` on a `PerlIO::scalar` handle now properly returns true when the file position is past the 2GB mark on 32-bit systems.

Attempting to write at file positions impossible for the platform now fail early rather than wrapping at 4GB.

- *Pod::Perldoc* has been upgraded to version 3.25.

  Filehandles opened for reading or writing now have `:encoding(UTF-8)` set. *[cpan #98019]*

- *POSIX* has been upgraded to version 1.53.

  The C99 math functions and constants (for example `acosh`, `isinf`, `isnan`, `round`, `trunc`; `M_E`, `M_SQRT2`, `M_PI`) have been added.

  `POSIX::tmpnam()` now produces a deprecation warning. [perl #122005]

- *Safe* has been upgraded to version 2.39.

  `reval` was not propagating void context properly.

- Scalar-List-Utils has been upgraded to version 1.41.

  A new module, *Sub::Util*, has been added, containing functions related to CODE refs, including `subname` (inspired by `Sub::Identity`) and `set_subname` (copied and renamed from `Sub::Name`). The use of `GetMagic` in `List::Util::reduce()` has also been fixed. *[cpan #63211]*

- *SDBM_File* has been upgraded to version 1.13.

  Simplified the build process. [perl #123413]

- *Time::Piece* has been upgraded to version 1.29.

  When pretty printing negative `Time::Seconds`, the "minus" is no longer lost.

- *Unicode::Collate* has been upgraded to version 1.12.

  Version 0.67's improved discontiguous contractions is invalidated by default and is supported as a parameter `long_contraction`.

- *Unicode::Normalize* has been upgraded to version 1.18.

  The XSUB implementation has been removed in favor of pure Perl.

- *Unicode::UCD* has been upgraded to version 0.61.

  A new function *property_values()* has been added to return a given property's possible values.

  A new function *charprop()* has been added to return the value of a given property for a given code point.

  A new function *charprops_all()* has been added to return the values of all Unicode properties for a given code point.

  A bug has been fixed so that *propaliases()* returns the correct short and long names for the Perl extensions where it was incorrect.

  A bug has been fixed so that *prop_value_aliases()* returns `undef` instead of a wrong result for properties that are Perl extensions.

  This module now works on EBCDIC platforms.

- *utf8* has been upgraded to version 1.17

  A mismatch between the documentation and the code in `utf8::downgrade()` was fixed in favor of the documentation. The optional second argument is now correctly treated as a perl boolean (true/false semantics) and not as an integer.

- *version* has been upgraded to version 0.9909.

  Numerous changes. See the *Changes* file in the CPAN distribution for details.

- *Win32* has been upgraded to version 0.51.

`GetOSName()` now supports Windows 8.1, and building in C++ mode now works.

- *Win32API::File* has been upgraded to version 0.1202

  Building in C++ mode now works.

- *XSLoader* has been upgraded to version 0.20.

  Allow XSLoader to load modules from a different namespace. [perl #122455]

Removed Modules and Pragmata

The following modules (and associated modules) have been removed from the core perl distribution:

- *CGI*
- *Module::Build*

Documentation

New Documentation

perlunicook

This document, by Tom Christiansen, provides examples of handling Unicode in Perl.

Changes to Existing Documentation

perlaix

- A note on long doubles has been added.

perlapi

- Note that `SvSetSV` doesn't do set magic.
- `sv_usepvn_flags` - fix documentation to mention the use of `Newx` instead of `malloc`.

  [perl #121869]
- Clarify where `NUL` may be embedded or is required to terminate a string.
- Some documentation that was previously missing due to formatting errors is now included.
- Entries are now organized into groups rather than by the file where they are found.
- Alphabetical sorting of entries is now done consistently (automatically by the POD generator) to make entries easier to find when scanning.

perldata

- The syntax of single-character variable names has been brought up-to-date and more fully explained.
- Hexadecimal floating point numbers are described, as are infinity and NaN.

perlebcdic

- This document has been significantly updated in the light of recent improvements to EBCDIC support.

perlfilter

- Added a *LIMITATIONS* section.

perlfunc

- Mention that `study()` is currently a no-op.
- Calling `delete` or `exists` on array values is now described as "strongly discouraged" rather than "deprecated".

- Improve documentation of `our`.
- `-l` now notes that it will return false if symlinks aren't supported by the file system. *[perl #121523]*
- Note that `exec LIST` and `system LIST` may fall back to the shell on Win32. Only the indirect-object syntax `exec PROGRAM LIST` and `system PROGRAM LIST` will reliably avoid using the shell.

  This has also been noted in *perlport*.

  [perl #122046]

perlguts

- The OOK example has been updated to account for COW changes and a change in the storage of the offset.
- Details on C level symbols and libperl.t added.
- Information on Unicode handling has been added
- Information on EBCDIC handling has been added

perlhack

- A note has been added about running on platforms with non-ASCII character sets
- A note has been added about performance testing

perlhacktips

- Documentation has been added illustrating the perils of assuming that there is no change to the contents of static memory pointed to by the return values of Perl's wrappers for C library functions.
- Replacements for `tmpfile`, `atoi`, `strtol`, and `strtoul` are now recommended.
- Updated documentation for the `test.valgrind` `make` target. *[perl #121431]*
- Information is given about writing test files portably to non-ASCII platforms.
- A note has been added about how to get a C language stack backtrace.

perlhpux

- Note that the message "Redeclaration of "sendpath" with a different storage class specifier" is harmless.

perllocale

- Updated for the enhancements in v5.22, along with some clarifications.

perlmodstyle

- Instead of pointing to the module list, we are now pointing to *PrePAN*.

perlop

- Updated for the enhancements in v5.22, along with some clarifications.

perlpodspec

- The specification of the pod language is changing so that the default encoding of pods that aren't in UTF-8 (unless otherwise indicated) is CP1252 instead of ISO 8859-1 (Latin1).

perlpolicy

- We now have a code of conduct for the *p5p* mailing list, as documented in *"STANDARDS OF CONDUCT" in perlpolicy*.

- The conditions for marking an experimental feature as non-experimental are now set out.
- Clarification has been made as to what sorts of changes are permissible in maintenance releases.

perlport

- Out-of-date VMS-specific information has been fixed and/or simplified.
- Notes about EBCDIC have been added.

perlre

- The description of the `/x` modifier has been clarified to note that comments cannot be continued onto the next line by escaping them; and there is now a list of all the characters that are considered whitespace by this modifier.
- The new `/n` modifier is described.
- A note has been added on how to make bracketed character class ranges portable to non-ASCII machines.

perlrebackslash

- Added documentation of `\b{sb}`, `\b{wb}`, `\b{gcb}`, and `\b{g}`.

perlrecharclass

- Clarifications have been added to *"Character Ranges" in perlrecharclass* to the effect `[A-Z]`, `[a-z]`, `[0-9]` and any subranges thereof in regular expression bracketed character classes are guaranteed to match exactly what a naive English speaker would expect them to match, even on platforms (such as EBCDIC) where perl has to do extra work to accomplish this.
- The documentation of Bracketed Character Classes has been expanded to cover the improvements in `qr/[\N{named sequence}]/` (see under *Selected Bug Fixes*).

perlref

- A new section has been added *Assigning to References*

perlsec

- Comments added on algorithmic complexity and tied hashes.

perlsyn

- An ambiguity in the documentation of the `...` statement has been corrected. *[perl #122661]*
- The empty conditional in `for` and `while` is now documented in *perlsyn*.

perlunicode

- This has had extensive revisions to bring it up-to-date with current Unicode support and to make it more readable. Notable is that Unicode 7.0 changed what it should do with non-characters. Perl retains the old way of handling for reasons of backward compatibility. See *"Noncharacter code points" in perlunicode*.

perluniintro

- Advice for how to make sure your strings and regular expression patterns are interpreted as Unicode has been updated.

perlvar

- `$]` is no longer listed as being deprecated. Instead, discussion has been added on the advantages and disadvantages of using it versus `$^V`.
- `${^ENCODING}` is now marked as deprecated.

- The entry for %^H has been clarified to indicate it can only handle simple values.

perlvms

- Out-of-date and/or incorrect material has been removed.
- Updated documentation on environment and shell interaction in VMS.

perlxs

- Added a discussion of locale issues in XS code.

Diagnostics

The following additions or changes have been made to diagnostic output, including warnings and fatal error messages. For the complete list of diagnostic messages, see *perldiag*.

New Diagnostics

New Errors

- *Bad symbol for scalar*

  (P) An internal request asked to add a scalar entry to something that wasn't a symbol table entry.

- *Can't use a hash as a reference*

  (F) You tried to use a hash as a reference, as in `%foo->{"bar"}` or `%$ref->{"hello"}`. Versions of perl <= 5.6.1 used to allow this syntax, but shouldn't have.

- *Can't use an array as a reference*

  (F) You tried to use an array as a reference, as in `@foo->[23]` or `@$ref->[99]`. Versions of perl <= 5.6.1 used to allow this syntax, but shouldn't have.

- *Can't use 'defined(@array)' (Maybe you should just omit the defined()?)*

  (F) `defined()` is not useful on arrays because it checks for an undefined *scalar* value. If you want to see if the array is empty, just use `if (@array) { # not empty }` for example.

- *Can't use 'defined(%hash)' (Maybe you should just omit the defined()?)*

  (F) `defined()` is not usually right on hashes.

  Although `defined %hash` is false on a plain not-yet-used hash, it becomes true in several non-obvious circumstances, including iterators, weak references, stash names, even remaining true after `undef %hash`. These things make `defined %hash` fairly useless in practice, so it now generates a fatal error.

  If a check for non-empty is what you wanted then just put it in boolean context (see *"Scalar values" in perldata*):

  ```
  if (%hash) {
     # not empty
  }
  ```

  If you had `defined %Foo::Bar::QUUX` to check whether such a package variable exists then that's never really been reliable, and isn't a good way to enquire about the features of a package, or whether it's loaded, etc.

- *Cannot chr %f*

  (F) You passed an invalid number (like an infinity or not-a-number) to `chr`.

- *Cannot compress %f in pack*

  (F) You tried converting an infinity or not-a-number to an unsigned character, which makes no sense.

- *Cannot pack %f with '%c'*

  (F) You tried converting an infinity or not-a-number to a character, which makes no sense.

- *Cannot print %f with '%c'*

  (F) You tried printing an infinity or not-a-number as a character (%c), which makes no sense. Maybe you meant `'%s'`, or just stringifying it?

- *charnames alias definitions may not contain a sequence of multiple spaces*

  (F) You defined a character name which had multiple space characters in a row. Change them to single spaces. Usually these names are defined in the `:alias` import argument to `use charnames`, but they could be defined by a translator installed into `$^H{charnames}`. See *"CUSTOM ALIASES" in charnames*.

- *charnames alias definitions may not contain trailing white-space*

  (F) You defined a character name which ended in a space character. Remove the trailing space(s). Usually these names are defined in the `:alias` import argument to `use charnames`, but they could be defined by a translator installed into `$^H{charnames}`. See *"CUSTOM ALIASES" in charnames*.

- *:const is not permitted on named subroutines*

  (F) The `const` attribute causes an anonymous subroutine to be run and its value captured at the time that it is cloned. Named subroutines are not cloned like this, so the attribute does not make sense on them.

- *Hexadecimal float: internal error*

  (F) Something went horribly bad in hexadecimal float handling.

- *Hexadecimal float: unsupported long double format*

  (F) You have configured Perl to use long doubles but the internals of the long double format are unknown, therefore the hexadecimal float output is impossible.

- *Illegal suidscript*

  (F) The script run under suidperl was somehow illegal.

- *In '(?...)', the '(' and '?' must be adjacent in regex; marked by <-- HERE in m/%s/*

  (F) The two-character sequence `"(?"` in this context in a regular expression pattern should be an indivisible token, with nothing intervening between the `"("` and the `"?"`, but you separated them.

- *In '(*VERB...)', the '(' and '*' must be adjacent in regex; marked by <-- HERE in m/%s/*

  (F) The two-character sequence `"(*"` in this context in a regular expression pattern should be an indivisible token, with nothing intervening between the `"("` and the `"*"`, but you separated them.

- *Invalid quantifier in {,} in regex; marked by <-- HERE in m/%s/*

  (F) The pattern looks like a {min,max} quantifier, but the min or max could not be parsed as a valid number: either it has leading zeroes, or it represents too big a number to cope with. The <-- HERE shows where in the regular expression the problem was discovered. See *perlre*.

- *'%s' is an unknown bound type in regex*

  (F) You used `\b{...}` or `\B{...}` and the `...` is not known to Perl. The current valid ones are given in *"\b{}, \b, \B{}, \B" in perlrebackslash*.

- *Missing or undefined argument to require*

  (F) You tried to call `require` with no argument or with an undefined value as an argument. `require` expects either a package name or a file-specification as an argument. See *"require"*

in perlfunc.

Formerly, `require` with no argument or `undef` warned about a Null filename.

New Warnings

- *\C is deprecated in regex*

  (D deprecated) The `/\C/` character class was deprecated in v5.20, and now emits a warning. It is intended that it will become an error in v5.24. This character class matches a single byte even if it appears within a multi-byte character, breaks encapsulation, and can corrupt UTF-8 strings.

- *"%s" is more clearly written simply as "%s" in regex; marked by <-- HERE in m/%s/*

  (W regexp) (only under `use re 'strict'` or within `(?[...])`)

  You specified a character that has the given plainer way of writing it, and which is also portable to platforms running with different character sets.

- *Argument "%s" treated as 0 in increment (++)*

  (W numeric) The indicated string was fed as an argument to the ++ operator which expects either a number or a string matching `/^[a-zA-Z]*[0-9]*\z/`. See *"Auto-increment and Auto-decrement" in perlop* for details.

- *Both or neither range ends should be Unicode in regex; marked by <-- HERE in m/%s/*

  (W regexp) (only under `use re 'strict'` or within `(?[...])`)

  In a bracketed character class in a regular expression pattern, you had a range which has exactly one end of it specified using `\N{}`, and the other end is specified using a non-portable mechanism. Perl treats the range as a Unicode range, that is, all the characters in it are considered to be the Unicode characters, and which may be different code points on some platforms Perl runs on. For example, `[\N{U+06}-\x08]` is treated as if you had instead said `[\N{U+06}-\N{U+08}]`, that is it matches the characters whose code points in Unicode are 6, 7, and 8. But that `\x08` might indicate that you meant something different, so the warning gets raised.

- *Can't do %s("%s") on non-UTF-8 locale; resolved to "%s".*

  (W locale) You are 1) running under "`use locale`"; 2) the current locale is not a UTF-8 one; 3) you tried to do the designated case-change operation on the specified Unicode character; and 4) the result of this operation would mix Unicode and locale rules, which likely conflict.

  The warnings category `locale` is new.

- *:const is experimental*

  (S experimental::const_attr) The `const` attribute is experimental. If you want to use the feature, disable the warning with `no warnings 'experimental::const_attr'`, but know that in doing so you are taking the risk that your code may break in a future Perl version.

- *gmtime(%f) failed*

  (W overflow) You called `gmtime` with a number that it could not handle: too large, too small, or NaN. The returned value is `undef`.

- *Hexadecimal float: exponent overflow*

  (W overflow) The hexadecimal floating point has larger exponent than the floating point supports.

- *Hexadecimal float: exponent underflow*

  (W overflow) The hexadecimal floating point has smaller exponent than the floating point supports.

- *Hexadecimal float: mantissa overflow*

(W overflow) The hexadecimal floating point literal had more bits in the mantissa (the part between the `0x` and the exponent, also known as the fraction or the significand) than the floating point supports.

- *Hexadecimal float: precision loss*

  (W overflow) The hexadecimal floating point had internally more digits than could be output. This can be caused by unsupported long double formats, or by 64-bit integers not being available (needed to retrieve the digits under some configurations).

- *Locale '%s' may not work well.%s*

  (W locale) You are using the named locale, which is a non-UTF-8 one, and which perl has determined is not fully compatible with what it can handle. The second `%s` gives a reason.

  The warnings category `locale` is new.

- *localtime(%f) failed*

  (W overflow) You called `localtime` with a number that it could not handle: too large, too small, or NaN. The returned value is `undef`.

- *Negative repeat count does nothing*

  (W numeric) You tried to execute the *x* repetition operator fewer than 0 times, which doesn't make sense.

- *NO-BREAK SPACE in a charnames alias definition is deprecated*

  (D deprecated) You defined a character name which contained a no-break space character. Change it to a regular space. Usually these names are defined in the `:alias` import argument to `use charnames`, but they could be defined by a translator installed into `$^H{charnames}`. See *"CUSTOM ALIASES" in charnames*.

- *Non-finite repeat count does nothing*

  (W numeric) You tried to execute the *x* repetition operator `Inf` (or `-Inf`) or NaN times, which doesn't make sense.

- *PerlIO layer ':win32' is experimental*

  (S experimental::win32_perlio) The `:win32` PerlIO layer is experimental. If you want to take the risk of using this layer, simply disable this warning:

  ```
  no warnings "experimental::win32_perlio";
  ```

- *Ranges of ASCII printables should be some subset of "0-9", "A-Z", or "a-z" in regex; marked by <-- HERE in m/%s/*

  (W regexp) (only under `use re 'strict'` or within `(?[...])`)

  Stricter rules help to find typos and other errors. Perhaps you didn't even intend a range here, if the `"-"` was meant to be some other character, or should have been escaped (like `"\-"`). If you did intend a range, the one that was used is not portable between ASCII and EBCDIC platforms, and doesn't have an obvious meaning to a casual reader.

  ```
  [3-7]    # OK; Obvious and portable
  [d-g]    # OK; Obvious and portable
  [A-Y]    # OK; Obvious and portable
  [A-z]    # WRONG; Not portable; not clear what is meant
  [a-Z]    # WRONG; Not portable; not clear what is meant
  [%-.]    # WRONG; Not portable; not clear what is meant
  [\x41-Z] # WRONG; Not portable; not obvious to non-geek
  ```

  (You can force portability by specifying a Unicode range, which means that the endpoints are specified by `\N{...}`, but the meaning may still not be obvious.) The stricter rules require

that ranges that start or stop with an ASCII character that is not a control have all their endpoints be a literal character, and not some escape sequence (like `"\x41"`), and the ranges must be all digits, or all uppercase letters, or all lowercase letters.

- *Ranges of digits should be from the same group in regex; marked by <-- HERE in m/%s/*

  (W regexp) (only under `use re 'strict'` or within `(?[...])`)

  Stricter rules help to find typos and other errors. You included a range, and at least one of the end points is a decimal digit. Under the stricter rules, when this happens, both end points should be digits in the same group of 10 consecutive digits.

- *Redundant argument in %s*

  (W redundant) You called a function with more arguments than were needed, as indicated by information within other arguments you supplied (*e.g.* a printf format). Currently only emitted when a printf-type format required fewer arguments than were supplied, but might be used in the future for *e.g. "pack" in perlfunc*.

  The warnings category `redundant` is new. See also *[perl #121025]*.

- *Replacement list is longer than search list*

  This is not a new diagnostic, but in earlier releases was accidentally not displayed if the transliteration contained wide characters. This is now fixed, so that you may see this diagnostic in places where you previously didn't (but should have).

- *Use of \b{} for non-UTF-8 locale is wrong. Assuming a UTF-8 locale*

  (W locale) You are matching a regular expression using locale rules, and a Unicode boundary is being matched, but the locale is not a Unicode one. This doesn't make sense. Perl will continue, assuming a Unicode (UTF-8) locale, but the results could well be wrong except if the locale happens to be ISO-8859-1 (Latin1) where this message is spurious and can be ignored.

  The warnings category `locale` is new.

- *Using /u for '%s' instead of /%s in regex; marked by <-- HERE in m/%s/*

  (W regexp) You used a Unicode boundary (`\b{...}` or `\B{...}`) in a portion of a regular expression where the character set modifiers `/a` or `/aa` are in effect. These two modifiers indicate an ASCII interpretation, and this doesn't make sense for a Unicode definition. The generated regular expression will compile so that the boundary uses all of Unicode. No other portion of the regular expression is affected.

- *The bitwise feature is experimental*

  (S experimental::bitwise) This warning is emitted if you use bitwise operators (`& | ^ ~ &. |. ^. ~.`) with the "bitwise" feature enabled. Simply suppress the warning if you want to use the feature, but know that in doing so you are taking the risk of using an experimental feature which may change or be removed in a future Perl version:

  ```
  no warnings "experimental::bitwise";
  use feature "bitwise";
  $x |.= $y;
  ```

- *Unescaped left brace in regex is deprecated, passed through in regex; marked by <-- HERE in m/%s/*

  (D deprecated, regexp) You used a literal `"{"` character in a regular expression pattern. You should change to use `"\{"` instead, because a future version of Perl (tentatively v5.26) will consider this to be a syntax error. If the pattern delimiters are also braces, any matching right brace (`"}"`) should also be escaped to avoid confusing the parser, for example,

  ```
  qr{abc\{def\}ghi}
  ```

- *Use of literal non-graphic characters in variable names is deprecated*

  (D deprecated) Using literal non-graphic (including control) characters in the source to refer to the *^FOO* variables, like `$^X` and `${^GLOBAL_PHASE}` is now deprecated.

- *Useless use of attribute "const"*

  (W misc) The `const` attribute has no effect except on anonymous closure prototypes. You applied it to a subroutine via *attributes.pm*. This is only useful inside an attribute handler for an anonymous subroutine.

- *Useless use of /d modifier in transliteration operator*

  This is not a new diagnostic, but in earlier releases was accidentally not displayed if the transliteration contained wide characters. This is now fixed, so that you may see this diagnostic in places where you previously didn't (but should have).

- *"use re 'strict'" is experimental*

  (S experimental::re_strict) The things that are different when a regular expression pattern is compiled under `'strict'` are subject to change in future Perl releases in incompatible ways; there are also proposals to change how to enable strict checking instead of using this subpragma. This means that a pattern that compiles today may not in a future Perl release. This warning is to alert you to that risk.

- *Warning: unable to close filehandle properly: %s*

  Warning: unable to close filehandle %s properly: %s

  (S io) Previously, perl silently ignored any errors when doing an implicit close of a filehandle, *i.e.* where the reference count of the filehandle reached zero and the user's code hadn't already called `close()`; *e.g.*

  ```
  {
      open my $fh, '>', $file  or die "open: '$file': $!\n";
      print $fh, $data  or die;
  } # implicit close here
  ```

  In a situation such as disk full, due to buffering, the error may only be detected during the final close, so not checking the result of the close is dangerous.

  So perl now warns in such situations.

- *Wide character (U+%X) in %s*

  (W locale) While in a single-byte locale (*i.e.*, a non-UTF-8 one), a multi-byte character was encountered. Perl considers this character to be the specified Unicode code point. Combining non-UTF-8 locales and Unicode is dangerous. Almost certainly some characters will have two different representations. For example, in the ISO 8859-7 (Greek) locale, the code point 0xC3 represents a Capital Gamma. But so also does 0x393. This will make string comparisons unreliable.

  You likely need to figure out how this multi-byte character got mixed up with your single-byte locale (or perhaps you thought you had a UTF-8 locale, but Perl disagrees).

  The warnings category `locale` is new.

Changes to Existing Diagnostics

- <> should be quotes

  This warning has been changed to *<> at require-statement should be quotes* to make the issue more identifiable.

- *Argument "%s" isn't numeric%s*

  The *perldiag* entry for this warning has added this clarifying note:

```
Note that for the Inf and NaN (infinity and not-a-number) the
definition of "numeric" is somewhat unusual: the strings themselves
(like "Inf") are considered numeric, and anything following them is
considered non-numeric.
```

- *Global symbol "%s" requires explicit package name*

  This message has had '(did you forget to declare "my %s"?)' appended to it, to make it more helpful to new Perl programmers. *[perl #121638]*

- '"my" variable &foo::bar can't be in a package' has been reworded to say 'subroutine' instead of 'variable'.

- *\N{} in character class restricted to one character in regex; marked by <-- HERE in m/%s/*

  This message has had *character class* changed to *inverted character class or as a range end-point is* to reflect improvements in `qr/[\N{named sequence}]/` (see under *Selected Bug Fixes*).

- *panic: frexp*

  This message has had ': `%f`' appended to it, to show what the offending floating point number is.

- *Possible precedence problem on bitwise %c operator* reworded as *Possible precedence problem on bitwise %s operator*.

- *Unsuccessful %s on filename containing newline*

  This warning is now only produced when the newline is at the end of the filename.

- "Variable %s will not stay shared" has been changed to say "Subroutine" when it is actually a lexical sub that will not stay shared.

- *Variable length lookbehind not implemented in regex m/%s/*

  The *perldiag* entry for this warning has had information about Unicode behavior added.

Diagnostic Removals

- "Ambiguous use of -foo resolved as -&foo()"

  There is actually no ambiguity here, and this impedes the use of negated constants; *e.g.*, `-Inf`.

- "Constant is not a FOO reference"

  Compile-time checking of constant dereferencing (*e.g.*, `my_constant->()`) has been removed, since it was not taking overloading into account. *[perl #69456] [perl #122607]*

Utility Changes

find2perl, s2p and a2p removal

- The *x2p/* directory has been removed from the Perl core.

  This removes find2perl, s2p and a2p. They have all been released to CPAN as separate distributions (`App::find2perl`, `App::s2p`, `App::a2p`).

h2ph

- *h2ph* now handles hexadecimal constants in the compiler's predefined macro definitions, as visible in `$Config{cppsymbols}`. *[perl #123784]*.

encguess

- No longer depends on non-core modules.

Configuration and Compilation

- *Configure* now checks for `lrintl()`, `lroundl()`, `llrintl()`, and `llroundl()`.
- *Configure* with `-Dmksymlinks` should now be faster. *[perl #122002]*.
- The `pthreads` and `cl` libraries will be linked by default if present. This allows XS modules that require threading to work on non-threaded perls. Note that you must still pass `-Dusethreads` if you want a threaded perl.
- For long doubles (to get more precision and range for floating point numbers) one can now use the GCC quadmath library which implements the quadruple precision floating point numbers on x86 and IA-64 platforms. See *INSTALL* for details.
- MurmurHash64A and MurmurHash64B can now be configured as the internal hash function.
- `make test.valgrind` now supports parallel testing.

  For example:

  ```
  TEST_JOBS=9 make test.valgrind
  ```

  See *"valgrind" in perlhacktips* for more information.

  [perl #121431]
- The MAD (Misc Attribute Decoration) build option has been removed

  This was an unmaintained attempt at preserving the Perl parse tree more faithfully so that automatic conversion of Perl 5 to Perl 6 would have been easier.

  This build-time configuration option had been unmaintained for years, and had probably seriously diverged on both Perl 5 and Perl 6 sides.
- A new compilation flag, `-DPERL_OP_PARENT` is available. For details, see the discussion below at *Internal Changes*.
- Pathtools no longer tries to load XS on miniperl. This speeds up building perl slightly.

Testing

- *t/porting/re_context.t* has been added to test that *utf8* and its dependencies only use the subset of the `$1..$n` capture vars that `Perl_save_re_context()` is hard-coded to localize, because that function has no efficient way of determining at runtime what vars to localize.
- Tests for performance issues have been added in the file *t/perf/taint.t*.
- Some regular expression tests are written in such a way that they will run very slowly if certain optimizations break. These tests have been moved into new files, *t/re/speed.t* and *t/re/speed_thr.t*, and are run with a `watchdog()`.
- `test.pl` now allows `plan skip_all => $reason`, to make it more compatible with `Test::More`.
- A new test script, *op/infnan.t*, has been added to test if infinity and NaN are working correctly. See *Infinity and NaN (not-a-number) handling improved*.

Platform Support

Regained Platforms

IRIX and Tru64 platforms are working again.

Some `make test` failures remain: *[perl #123977]* and *[perl #125298]* for IRIX; *[perl #124212]*, *[cpan #99605]*, and *[cpan #104836* for Tru64.

z/OS running EBCDIC Code Page 1047

Core perl now works on this EBCDIC platform. Earlier perls also worked, but, even though support wasn't officially withdrawn, recent perls would not compile and run well. Perl 5.20 would work, but had many bugs which have now been fixed. Many CPAN modules that ship with Perl still fail tests, including `Pod::Simple`. However the version of `Pod::Simple` currently on CPAN should work; it was fixed too late to include in Perl 5.22. Work is under way to fix many of the still-broken CPAN modules, which likely will be installed on CPAN when completed, so that you may not have to wait until Perl 5.24 to get a working version.

Discontinued Platforms

NeXTSTEP/OPENSTEP

NeXTSTEP was a proprietary operating system bundled with NeXT's workstations in the early to mid 90s; OPENSTEP was an API specification that provided a NeXTSTEP-like environment on a non-NeXTSTEP system. Both are now long dead, so support for building Perl on them has been removed.

Platform-Specific Notes

EBCDIC

Special handling is required of the perl interpreter on EBCDIC platforms to get `qr/[i-j]/` to match only `"i"` and `"j"`, since there are 7 characters between the code points for `"i"` and `"j"`. This special handling had only been invoked when both ends of the range are literals. Now it is also invoked if any of the `\N{...}` forms for specifying a character by name or Unicode code point is used instead of a literal. See *"Character Ranges" in perlrecharclass*.

HP-UX

The archname now distinguishes use64bitint from use64bitall.

Android

Build support has been improved for cross-compiling in general and for Android in particular.

VMS

- When spawning a subprocess without waiting, the return value is now the correct PID.
- Fix a prototype so linking doesn't fail under the VMS C++ compiler.
- `finite`, `finitel`, and `isfinite` detection has been added to `configure.com`, environment handling has had some minor changes, and a fix for legacy feature checking status.

Win32

- *miniperl.exe* is now built with `-fno-strict-aliasing`, allowing 64-bit builds to complete on GCC 4.8. *[perl #123976]*
- `nmake minitest` now works on Win32. Due to dependency issues you need to build `nmake test-prep` first, and a small number of the tests fail. *[perl #123394]*
- Perl can now be built in C++ mode on Windows by setting the makefile macro `USE_CPLUSPLUS` to the value "define".
- The list form of piped open has been implemented for Win32. Note: unlike `system LIST` this does not fall back to the shell. *[perl #121159]*
- New `DebugSymbols` and `DebugFull` configuration options added to Windows makefiles.
- Previously, compiling XS modules (including CPAN ones) using Visual C++ for Win64 resulted in around a dozen warnings per file from *hv_func.h*. These warnings have been silenced.

- Support for building without PerlIO has been removed from the Windows makefiles. Non-PerlIO builds were all but deprecated in Perl 5.18.0 and are already not supported by *Configure* on POSIX systems.
- Between 2 and 6 milliseconds and seven I/O calls have been saved per attempt to open a perl module for each path in `@INC`.
- Intel C builds are now always built with C99 mode on.
- `%I64d` is now being used instead of `%lld` for MinGW.
- In the experimental `:win32` layer, a crash in `open` was fixed. Also opening */dev/null* (which works under Win32 Perl's default `:unix` layer) was implemented for `:win32`. *[perl #122224]*
- A new makefile option, `USE_LONG_DOUBLE`, has been added to the Windows dmake makefile for gcc builds only. Set this to "define" if you want perl to use long doubles to give more accuracy and range for floating point numbers.

OpenBSD

On OpenBSD, Perl will now default to using the system `malloc` due to the security features it provides. Perl's own malloc wrapper has been in use since v5.14 due to performance reasons, but the OpenBSD project believes the tradeoff is worth it and would prefer that users who need the speed specifically ask for it.

[perl #122000].

Solaris

- We now look for the Sun Studio compiler in both */opt/solstudio** and */opt/solarisstudio**.
- Builds on Solaris 10 with `-Dusedtrace` would fail early since make didn't follow implied dependencies to build `perldtrace.h`. Added an explicit dependency to `depend`. *[perl #120120]*
- C99 options have been cleaned up; hints look for `solstudio` as well as `SUNWspro`; and support for native `setenv` has been added.

Internal Changes

- Experimental support has been added to allow ops in the optree to locate their parent, if any. This is enabled by the non-default build option `-DPERL_OP_PARENT`. It is envisaged that this will eventually become enabled by default, so XS code which directly accesses the `op_sibling` field of ops should be updated to be future-proofed.

  On `PERL_OP_PARENT` builds, the `op_sibling` field has been renamed `op_sibparent` and a new flag, `op_moresib`, added. On the last op in a sibling chain, `op_moresib` is false and `op_sibparent` points to the parent (if any) rather than being `NULL`.

  To make existing code work transparently whether using `PERL_OP_PARENT` or not, a number of new macros and functions have been added that should be used, rather than directly manipulating `op_sibling`.

  For the case of just reading `op_sibling` to determine the next sibling, two new macros have been added. A simple scan through a sibling chain like this:

  ```
  for (; kid->op_sibling; kid = kid->op_sibling) { ... }
  ```

  should now be written as:

  ```
  for (; OpHAS_SIBLING(kid); kid = OpSIBLING(kid)) { ... }
  ```

  For altering optrees, a general-purpose function `op_sibling_splice()` has been added, which allows for manipulation of a chain of sibling ops. By analogy with the Perl function

`splice()`, it allows you to cut out zero or more ops from a sibling chain and replace them with zero or more new ops. It transparently handles all the updating of sibling, parent, op_last pointers etc.

If you need to manipulate ops at a lower level, then three new macros, `OpMORESIB_set`, `OpLASTSIB_set` and `OpMAYBESIB_set` are intended to be a low-level portable way to set `op_sibling` / `op_sibparent` while also updating `op_moresib`. The first sets the sibling pointer to a new sibling, the second makes the op the last sibling, and the third conditionally does the first or second action. Note that unlike `op_sibling_splice()` these macros won't maintain consistency in the parent at the same time (*e.g.* by updating `op_first` and `op_last` where appropriate).

A C-level `Perl_op_parent()` function and a Perl-level `B::OP::parent()` method have been added. The C function only exists under `PERL_OP_PARENT` builds (using it is build-time error on vanilla perls). `B::OP::parent()` exists always, but on a vanilla build it always returns `NULL`. Under `PERL_OP_PARENT`, they return the parent of the current op, if any. The variable `$B::OP::does_parent` allows you to determine whether `B` supports retrieving an op's parent.

`PERL_OP_PARENT` was introduced in 5.21.2, but the interface was changed considerably in 5.21.11. If you updated your code before the 5.21.11 changes, it may require further revision. The main changes after 5.21.2 were:

- The `OP_SIBLING` and `OP_HAS_SIBLING` macros have been renamed `OpSIBLING` and `OpHAS_SIBLING` for consistency with other op-manipulating macros.
- The `op_lastsib` field has been renamed `op_moresib`, and its meaning inverted.
- The macro `OpSIBLING_set` has been removed, and has been superseded by `OpMORESIB_set` *et al*.
- The `op_sibling_splice()` function now accepts a null `parent` argument where the splicing doesn't affect the first or last ops in the sibling chain

- Macros have been created to allow XS code to better manipulate the POSIX locale category `LC_NUMERIC`. See *"Locale-related functions and macros" in perlapi*.
- The previous `atoi` *et al* replacement function, `grok_atou`, has now been superseded by `grok_atoUV`. See *perlclib* for details.
- A new function, `Perl_sv_get_backrefs()`, has been added which allows you retrieve the weak references, if any, which point at an SV.
- The `screaminstr()` function has been removed. Although marked as public API, it was undocumented and had no usage in CPAN modules. Calling it has been fatal since 5.17.0.
- The `newDEFSVOP()`, `block_start()`, `block_end()` and `intro_my()` functions have been added to the API.
- The internal `convert` function in *op.c* has been renamed `op_convert_list` and added to the API.
- The `sv_magic()` function no longer forbids "ext" magic on read-only values. After all, perl can't know whether the custom magic will modify the SV or not. *[perl #123103]*.
- Accessing *"CvPADLIST" in perlapi* on an XSUB is now forbidden.

  The `CvPADLIST` field has been reused for a different internal purpose for XSUBs. So in particular, you can no longer rely on it being NULL as a test of whether a CV is an XSUB. Use `CvISXSUB()` instead.
- SVs of type `SVt_NV` are now sometimes bodiless when the build configuration and platform allow it: specifically, when `sizeof(NV) <= sizeof(IV)`. "Bodiless" means that the NV

value is stored directly in the head of an SV, without requiring a separate body to be allocated. This trick has already been used for IVs since 5.9.2 (though in the case of IVs, it is always used, regardless of platform and build configuration).

- The `$DB::single`, `$DB::signal` and `$DB::trace` variables now have set- and get-magic that stores their values as IVs, and those IVs are used when testing their values in `pp_dbstate()`. This prevents perl from recursing infinitely if an overloaded object is assigned to any of those variables. *[perl #122445]*.

- `Perl_tmps_grow()`, which is marked as public API but is undocumented, has been removed from the public API. This change does not affect XS code that uses the `EXTEND_MORTAL` macro to pre-extend the mortal stack.

- Perl's internals no longer sets or uses the `SVs_PADMY` flag. `SvPADMY()` now returns a true value for anything not marked `PADTMP` and `SVs_PADMY` is now defined as 0.

- The macros `SETsv` and `SETsvUN` have been removed. They were no longer used in the core since commit 6f1401dc2a five years ago, and have not been found present on CPAN.

- The `SvFAKE` bit (unused on HVs) got informally reserved by David Mitchell for future work on vtables.

- The `sv_catpvn_flags()` function accepts `SV_CATBYTES` and `SV_CATUTF8` flags, which specify whether the appended string is bytes or UTF-8, respectively. (These flags have in fact been present since 5.16.0, but were formerly not regarded as part of the API.)

- A new opcode class, `METHOP`, has been introduced. It holds information used at runtime to improve the performance of class/object method calls.

  `OP_METHOD` and `OP_METHOD_NAMED` have changed from being `UNOP/SVOP` to being `METHOP`.

- `cv_name()` is a new API function that can be passed a CV or GV. It returns an SV containing the name of the subroutine, for use in diagnostics.

  [perl #116735] [perl #120441]

- `cv_set_call_checker_flags()` is a new API function that works like `cv_set_call_checker()`, except that it allows the caller to specify whether the call checker requires a full GV for reporting the subroutine's name, or whether it could be passed a CV instead. Whatever value is passed will be acceptable to `cv_name()`. `cv_set_call_checker()` guarantees there will be a GV, but it may have to create one on the fly, which is inefficient. *[perl #116735]*

- `CvGV` (which is not part of the API) is now a more complex macro, which may call a function and reify a GV. For those cases where it has been used as a boolean, `CvHASGV` has been added, which will return true for CVs that notionally have GVs, but without reifying the GV. `CvGV` also returns a GV now for lexical subs. *[perl #120441]*

- The *"sync_locale" in perlapi* function has been added to the public API. Changing the program's locale should be avoided by XS code. Nevertheless, certain non-Perl libraries called from XS need to do so, such as `Gtk`. When this happens, Perl needs to be told that the locale has changed. Use this function to do so, before returning to Perl.

- The defines and labels for the flags in the `op_private` field of OPs are now auto-generated from data in *regen/op_private*. The noticeable effect of this is that some of the flag output of `Concise` might differ slightly, and the flag output of `perl -Dx` may differ considerably (they both use the same set of labels now). Also, debugging builds now have a new assertion in `op_free()` to ensure that the op doesn't have any unrecognized flags set in `op_private`.

- The deprecated variable `PL_sv_objcount` has been removed.

- Perl now tries to keep the locale category `LC_NUMERIC` set to "C" except around operations that need it to be set to the program's underlying locale. This protects the many XS modules that cannot cope with the decimal radix character not being a dot. Prior to this release, Perl initialized this category to "C", but a call to `POSIX::setlocale()` would change it. Now such a call will change the underlying locale of the `LC_NUMERIC` category for the program, but the locale exposed to XS code will remain "C". There are new macros to manipulate the LC_NUMERIC locale, including `STORE_LC_NUMERIC_SET_TO_NEEDED` and `STORE_LC_NUMERIC_FORCE_TO_UNDERLYING`. See *"Locale-related functions and macros" in perlapi*.

- A new macro *`isUTF8_CHAR`* has been written which efficiently determines if the string given by its parameters begins with a well-formed UTF-8 encoded character.

- The following private API functions had their context parameter removed: `Perl_cast_ulong`, `Perl_cast_i32`, `Perl_cast_iv`, `Perl_cast_uv`, `Perl_cv_const_sv`, `Perl_mg_find`, `Perl_mg_findext`, `Perl_mg_magical`, `Perl_mini_mktime`, `Perl_my_dirfd`, `Perl_sv_backoff`, `Perl_utf8_hop`.

  Note that the prefix-less versions of those functions that are part of the public API, such as `cast_i32()`, remain unaffected.

- The `PADNAME` and `PADNAMELIST` types are now separate types, and no longer simply aliases for SV and AV. *[perl #123223]*.

- Pad names are now always UTF-8. The `PadnameUTF8` macro always returns true. Previously, this was effectively the case already, but any support for two different internal representations of pad names has now been removed.

- A new op class, `UNOP_AUX`, has been added. This is a subclass of `UNOP` with an `op_aux` field added, which points to an array of unions of UV, SV* etc. It is intended for where an op needs to store more data than a simple `op_sv` or whatever. Currently the only op of this type is `OP_MULTIDEREF` (see next item).

- A new op has been added, `OP_MULTIDEREF`, which performs one or more nested array and hash lookups where the key is a constant or simple variable. For example the expression `$a[0]{$k}[$i]`, which previously involved ten `rv2Xv`, `Xelem`, `gvsv` and `const` ops is now performed by a single `multideref` op. It can also handle `local`, `exists` and `delete`. A non-simple index expression, such as `[$i+1]` is still done using `aelem`/`helem`, and single-level array lookup with a small constant index is still done using `aelemfast`.

Selected Bug Fixes

- `close` now sets `$!`

  When an I/O error occurs, the fact that there has been an error is recorded in the handle. `close` returns false for such a handle. Previously, the value of `$!` would be untouched by `close`, so the common convention of writing `close $fh or die $!` did not work reliably. Now the handle records the value of `$!`, too, and `close` restores it.

- `no re` now can turn off everything that `use re` enables

  Previously, running `no re` would turn off only a few things. Now it can turn off all the enabled things. For example, the only way to stop debugging, once enabled, was to exit the enclosing block; that is now fixed.

- `pack("D", $x)` and `pack("F", $x)` now zero the padding on x86 long double builds. Under some build options on GCC 4.8 and later, they used to either overwrite the zero-initialized padding, or bypass the initialized buffer entirely. This caused *op/pack.t* to fail. *[perl #123971]*

- Extending an array cloned from a parent thread could result in "Modification of a read-only value attempted" errors when attempting to modify the new elements. *[perl #124127]*

- An assertion failure and subsequent crash with `*x=<y>` has been fixed. *[perl #123790]*
- A possible crashing/looping bug related to compiling lexical subs has been fixed. *[perl #124099]*
- UTF-8 now works correctly in function names, in unquoted HERE-document terminators, and in variable names used as array indexes. *[perl #124113]*
- Repeated global pattern matches in scalar context on large tainted strings were exponentially slow depending on the current match position in the string. *[perl #123202]*
- Various crashes due to the parser getting confused by syntax errors have been fixed. *[perl #123801] [perl #123802] [perl #123955] [perl #123995]*
- `split` in the scope of lexical `$_` has been fixed not to fail assertions. *[perl #123763]*
- `my $x : attr` syntax inside various list operators no longer fails assertions. *[perl #123817]*
- An @ sign in quotes followed by a non-ASCII digit (which is not a valid identifier) would cause the parser to crash, instead of simply trying the @ as literal. This has been fixed. *[perl #123963]*
- `*bar::=*foo::=*glob_with_hash` has been crashing since Perl 5.14, but no longer does. *[perl #123847]*
- `foreach` in scalar context was not pushing an item on to the stack, resulting in bugs. (`print 4, scalar do { foreach(@x){} } + 1` would print 5.) It has been fixed to return `undef`. *[perl #124004]*
- Several cases of data used to store environment variable contents in core C code being potentially overwritten before being used have been fixed. *[perl #123748]*
- Some patterns starting with `/.*..../` matched against long strings have been slow since v5.8, and some of the form `/.*..../i` have been slow since v5.18. They are now all fast again. *[perl #123743]*.
- The original visible value of `$/` is now preserved when it is set to an invalid value. Previously if you set `$/` to a reference to an array, for example, perl would produce a runtime error and not set `PL_rs`, but Perl code that checked `$/` would see the array reference. *[perl #123218]*.
- In a regular expression pattern, a POSIX class, like `[:ascii:]`, must be inside a bracketed character class, like `qr/[[:ascii:]]/`. A warning is issued when something looking like a POSIX class is not inside a bracketed class. That warning wasn't getting generated when the POSIX class was negated: `[:^ascii:]`. This is now fixed.
- Perl 5.14.0 introduced a bug whereby `eval { LABEL: }` would crash. This has been fixed. *[perl #123652]*.
- Various crashes due to the parser getting confused by syntax errors have been fixed. *[perl #123617]*. *[perl #123737]*. *[perl #123753]*. *[perl #123677]*.
- Code like `/$a[/` used to read the next line of input and treat it as though it came immediately after the opening bracket. Some invalid code consequently would parse and run, but some code caused crashes, so this is now disallowed. *[perl #123712]*.
- Fix argument underflow for `pack`. *[perl #123874]*.
- Fix handling of non-strict `\x{}`. Now `\x{}` is equivalent to `\x{0}` instead of faulting.
- `stat -t` is now no longer treated as stackable, just like `-t stat`. *[perl #123816]*.
- The following no longer causes a SEGV: `qr{x+(y(?0))*}`.

- Fixed infinite loop in parsing backrefs in regexp patterns.
- Several minor bug fixes in behavior of Infinity and NaN, including warnings when stringifying Infinity-like or NaN-like strings. For example, "NaNcy" doesn't numify to NaN anymore.
- A bug in regular expression patterns that could lead to segfaults and other crashes has been fixed. This occurred only in patterns compiled with `/i` while taking into account the current POSIX locale (which usually means they have to be compiled within the scope of `use locale`), and there must be a string of at least 128 consecutive bytes to match. *[perl #123539]*.
- `s///g` now works on very long strings (where there are more than 2 billion iterations) instead of dying with 'Substitution loop'. *[perl #103260]*. *[perl #123071]*.
- `gmtime` no longer crashes with not-a-number values. *[perl #123495]*.
- `\()` (a reference to an empty list), and `y///` with lexical `$_` in scope, could both do a bad write past the end of the stack. They have both been fixed to extend the stack first.
- `prototype()` with no arguments used to read the previous item on the stack, so `print "foo", prototype()` would print foo's prototype. It has been fixed to infer `$_` instead. *[perl #123514]*.
- Some cases of lexical state subs declared inside predeclared subs could crash, for example when evalling a string including the name of an outer variable, but no longer do.
- Some cases of nested lexical state subs inside anonymous subs could cause 'Bizarre copy' errors or possibly even crashes.
- When trying to emit warnings, perl's default debugger (*perl5db.pl*) was sometimes giving 'Undefined subroutine &DB::db_warn called' instead. This bug, which started to occur in Perl 5.18, has been fixed. *[perl #123553]*.
- Certain syntax errors in substitutions, such as `s/${<>{})//`, would crash, and had done so since Perl 5.10. (In some cases the crash did not start happening till 5.16.) The crash has, of course, been fixed. *[perl #123542]*.
- Fix a couple of string grow size calculation overflows; in particular, a repeat expression like `33 x ~3` could cause a large buffer overflow since the new output buffer size was not correctly handled by `SvGROW()`. An expression like this now properly produces a memory wrap panic. *[perl #123554]*.
- `formline("@...", "a");` would crash. The `FF_CHECKNL` case in `pp_formline()` didn't set the pointer used to mark the chop position, which led to the `FF_MORE` case crashing with a segmentation fault. This has been fixed. *[perl #123538]*.
- A possible buffer overrun and crash when parsing a literal pattern during regular expression compilation has been fixed. *[perl #123604]*.
- `fchmod()` and `futimes()` now set `$!` when they fail due to being passed a closed file handle. *[perl #122703]*.
- `op_free()` and `scalarvoid()` no longer crash due to a stack overflow when freeing a deeply recursive op tree. *[perl #108276]*.
- In Perl 5.20.0, `$^N` accidentally had the internal UTF-8 flag turned off if accessed from a code block within a regular expression, effectively UTF-8-encoding the value. This has been fixed. *[perl #123135]*.
- A failed `semctl` call no longer overwrites existing items on the stack, which means that `(semctl(-1,0,0,0))[0]` no longer gives an "uninitialized" warning.

- `else{foo()}` with no space before `foo` is now better at assigning the right line number to that statement. *[perl #122695]*.

- Sometimes the assignment in `@array = split` gets optimised so that `split` itself writes directly to the array. This caused a bug, preventing this assignment from being used in lvalue context. So `(@a=split//,"foo")=bar()` was an error. (This bug probably goes back to Perl 3, when the optimisation was added.) It has now been fixed. *[perl #123057]*.

- When an argument list fails the checks specified by a subroutine signature (which is still an experimental feature), the resulting error messages now give the file and line number of the caller, not of the called subroutine. *[perl #121374]*.

- The flip-flop operators (`..` and `...` in scalar context) used to maintain a separate state for each recursion level (the number of times the enclosing sub was called recursively), contrary to the documentation. Now each closure has one internal state for each flip-flop. *[perl #122829]*.

- The flip-flop operator (`..` in scalar context) would return the same scalar each time, unless the containing subroutine was called recursively. Now it always returns a new scalar. *[perl #122829]*.

- `use`, `no`, statement labels, special blocks (`BEGIN`) and pod are now permitted as the first thing in a `map` or `grep` block, the block after `print` or `say` (or other functions) returning a handle, and within `${...}`, `@{...}`, etc. *[perl #122782]*.

- The repetition operator `x` now propagates lvalue context to its left-hand argument when used in contexts like `foreach`. That allows `for((@#that_array)x2) { ... }` to work as expected if the loop modifies `$_`.

- `(...) x ...` in scalar context used to corrupt the stack if one operand was an object with "x" overloading, causing erratic behavior. *[perl #121827]*.

- Assignment to a lexical scalar is often optimised away; for example in `my $x; $x = $y + $z`, the assign operator is optimised away and the add operator writes its result directly to `$x`. Various bugs related to this optimisation have been fixed. Certain operators on the right-hand side would sometimes fail to assign the value at all or assign the wrong value, or would call STORE twice or not at all on tied variables. The operators affected were `$foo++`, `$foo--`, and `-$foo` under `use integer`, `chomp`, `chr` and `setpgrp`.

- List assignments were sometimes buggy if the same scalar ended up on both sides of the assignment due to use of `tied`, `values` or `each`. The result would be the wrong value getting assigned.

- `setpgrp($nonzero)` (with one argument) was accidentally changed in 5.16 to mean `setpgrp(0)`. This has been fixed.

- `__SUB__` could return the wrong value or even corrupt memory under the debugger (the `-d` switch) and in subs containing `eval $string`.

- When `sub () { $var }` becomes inlinable, it now returns a different scalar each time, just as a non-inlinable sub would, though Perl still optimises the copy away in cases where it would make no observable difference.

- `my sub f () { $var }` and `sub () : attr { $var }` are no longer eligible for inlining. The former would crash; the latter would just throw the attributes away. An exception is made for the little-known `:method` attribute, which does nothing much.

- Inlining of subs with an empty prototype is now more consistent than before. Previously, a sub with multiple statements, of which all but the last were optimised away, would be inlinable only if it were an anonymous sub containing a string `eval` or `state` declaration or closing over an outer lexical variable (or any anonymous sub under the debugger). Now any sub that gets

folded to a single constant after statements have been optimised away is eligible for inlining. This applies to things like `sub () { jabber() if DEBUG; 42 }`.

Some subroutines with an explicit `return` were being made inlinable, contrary to the documentation, Now `return` always prevents inlining.

- On some systems, such as VMS, `crypt` can return a non-ASCII string. If a scalar assigned to had contained a UTF-8 string previously, then `crypt` would not turn off the UTF-8 flag, thus corrupting the return value. This would happen with `$lexical = crypt ...`.
- `crypt` no longer calls `FETCH` twice on a tied first argument.
- An unterminated here-doc on the last line of a quote-like operator (`qq[${ <<END }]`, `/(?{ <<END })/`) no longer causes a double free. It started doing so in 5.18.
- `index()` and `rindex()` no longer crash when used on strings over 2GB in size. *[perl #121562]*.
- A small, previously intentional, memory leak in `PERL_SYS_INIT`/`PERL_SYS_INIT3` on Win32 builds was fixed. This might affect embedders who repeatedly create and destroy perl engines within the same process.
- `POSIX::localeconv()` now returns the data for the program's underlying locale even when called from outside the scope of `use locale`.
- `POSIX::localeconv()` now works properly on platforms which don't have `LC_NUMERIC` and/or `LC_MONETARY`, or for which Perl has been compiled to disregard either or both of these locale categories. In such circumstances, there are now no entries for the corresponding values in the hash returned by `localeconv()`.
- `POSIX::localeconv()` now marks appropriately the values it returns as UTF-8 or not. Previously they were always returned as bytes, even if they were supposed to be encoded as UTF-8.
- On Microsoft Windows, within the scope of `use locale`, the following POSIX character classes gave results for many locales that did not conform to the POSIX standard: `[[:alnum:]]`, `[[:alpha:]]`, `[[:blank:]]`, `[[:digit:]]`, `[[:graph:]]`, `[[:lower:]]`, `[[:print:]]`, `[[:punct:]]`, `[[:upper:]]`, `[[:word:]]`, and `[[:xdigit:]]`. This was because the underlying Microsoft implementation does not follow the standard. Perl now takes special precautions to correct for this.
- Many issues have been detected by *Coverity* and fixed.
- `system()` and friends should now work properly on more Android builds.

  Due to an oversight, the value specified through `-Dtargetsh` to *Configure* would end up being ignored by some of the build process. This caused perls cross-compiled for Android to end up with defective versions of `system()`, `exec()` and backticks: the commands would end up looking for `/bin/sh` instead of `/system/bin/sh`, and so would fail for the vast majority of devices, leaving `$!` as `ENOENT`.
- `qr(...\(...\)...)`, `qr[...\[...\]...]`, and `qr{...\{...\}...}` now work. Previously it was impossible to escape these three left-characters with a backslash within a regular expression pattern where otherwise they would be considered metacharacters, and the pattern opening delimiter was the character, and the closing delimiter was its mirror character.
- `s///e` on tainted UTF-8 strings corrupted `pos()`. This bug, introduced in 5.20, is now fixed. *[perl #122148]*.
- A non-word boundary in a regular expression (`\B`) did not always match the end of the string; in particular `q{} =~ /\B/` did not match. This bug, introduced in perl 5.14, is now fixed. *[perl*

#122090].

- `" P" =~ /(?=.*P)P/` should match, but did not. This is now fixed. *[perl #122171]*.
- Failing to compile `use Foo` in an `eval` could leave a spurious `BEGIN` subroutine definition, which would produce a "Subroutine BEGIN redefined" warning on the next use of `use`, or other `BEGIN` block. *[perl #122107]*.
- `method { BLOCK } ARGS` syntax now correctly parses the arguments if they begin with an opening brace. *[perl #46947]*.
- External libraries and Perl may have different ideas of what the locale is. This is problematic when parsing version strings if the locale's numeric separator has been changed. Version parsing has been patched to ensure it handles the locales correctly. *[perl #121930]*.
- A bug has been fixed where zero-length assertions and code blocks inside of a regex could cause `pos` to see an incorrect value. *[perl #122460]*.
- Dereferencing of constants now works correctly for typeglob constants. Previously the glob was stringified and its name looked up. Now the glob itself is used. *[perl #69456]*
- When parsing a sigil (`$ @ % &`) followed by braces, the parser no longer tries to guess whether it is a block or a hash constructor (causing a syntax error when it guesses the latter), since it can only be a block.
- `undef $reference` now frees the referent immediately, instead of hanging on to it until the next statement. *[perl #122556]*
- Various cases where the name of a sub is used (autoload, overloading, error messages) used to crash for lexical subs, but have been fixed.
- Bareword lookup now tries to avoid vivifying packages if it turns out the bareword is not going to be a subroutine name.
- Compilation of anonymous constants (*e.g.*, `sub () { 3 }`) no longer deletes any subroutine named `__ANON__` in the current package. Not only was `*__ANON__{CODE}` cleared, but there was a memory leak, too. This bug goes back to Perl 5.8.0.
- Stub declarations like `sub f;` and `sub f ();` no longer wipe out constants of the same name declared by `use constant`. This bug was introduced in Perl 5.10.0.
- `qr/[\N{named sequence}]/` now works properly in many instances.

  Some names known to `\N{...}` refer to a sequence of multiple characters, instead of the usual single character. Bracketed character classes generally only match single characters, but now special handling has been added so that they can match named sequences, but not if the class is inverted or the sequence is specified as the beginning or end of a range. In these cases, the only behavior change from before is a slight rewording of the fatal error message given when this class is part of a `?[...]`) construct. When the `[...]` stands alone, the same non-fatal warning as before is raised, and only the first character in the sequence is used, again just as before.
- Tainted constants evaluated at compile time no longer cause unrelated statements to become tainted. *[perl #122669]*
- `open $$fh, ...`, which vivifies a handle with a name like `"main::_GEN_0"`, was not giving the handle the right reference count, so a double free could happen.
- When deciding that a bareword was a method name, the parser would get confused if an `our` sub with the same name existed, and look up the method in the package of the `our` sub, instead of the package of the invocant.

- The parser no longer gets confused by `\U=` within a double-quoted string. It used to produce a syntax error, but now compiles it correctly. *[perl #80368]*
- It has always been the intention for the `-B` and `-T` file test operators to treat UTF-8 encoded files as text. (*perlfunc* has been updated to say this.) Previously, it was possible for some files to be considered UTF-8 that actually weren't valid UTF-8. This is now fixed. The operators now work on EBCDIC platforms as well.
- Under some conditions warning messages raised during regular expression pattern compilation were being output more than once. This has now been fixed.
- Perl 5.20.0 introduced a regression in which a UTF-8 encoded regular expression pattern that contains a single ASCII lowercase letter did not match its uppercase counterpart. That has been fixed in both 5.20.1 and 5.22.0. *[perl #122655]*
- Constant folding could incorrectly suppress warnings if lexical warnings (`use warnings` or `no warnings`) were not in effect and `$^W` were false at compile time and true at run time.
- Loading Unicode tables during a regular expression match could cause assertion failures under debugging builds if the previous match used the very same regular expression. *[perl #122747]*
- Thread cloning used to work incorrectly for lexical subs, possibly causing crashes or double frees on exit.
- Since Perl 5.14.0, deleting `$SomePackage::{__ANON__}` and then undefining an anonymous subroutine could corrupt things internally, resulting in *Devel::Peek* crashing or *B.pm* giving nonsensical data. This has been fixed.
- `(caller $n)[3]` now reports names of lexical subs, instead of treating them as `"(unknown)"`.
- `sort subname LIST` now supports using a lexical sub as the comparison routine.
- Aliasing (*e.g.*, via `*x = *y`) could confuse list assignments that mention the two names for the same variable on either side, causing wrong values to be assigned. *[perl #15667]*
- Long here-doc terminators could cause a bad read on short lines of input. This has been fixed. It is doubtful that any crash could have occurred. This bug goes back to when here-docs were introduced in Perl 3.000 twenty-five years ago.
- An optimization in `split` to treat `split /^/` like `split /^/m` had the unfortunate side-effect of also treating `split /\A/` like `split /^/m`, which it should not. This has been fixed. (Note, however, that `split /^x/` does not behave like `split /^x/m`, which is also considered to be a bug and will be fixed in a future version.) *[perl #122761]*
- The little-known `my Class $var` syntax (see *fields* and *attributes*) could get confused in the scope of `use utf8` if `Class` were a constant whose value contained Latin-1 characters.
- Locking and unlocking values via *Hash::Util* or `Internals::SvREADONLY` no longer has any effect on values that were read-only to begin with. Previously, unlocking such values could result in crashes, hangs or other erratic behavior.
- Some unterminated `(?(...)...)` constructs in regular expressions would either crash or give erroneous error messages. `/(?(1)/` is one such example.
- `pack "w", $tied` no longer calls FETCH twice.
- List assignments like `($x, $z) = (1, $y)` now work correctly if `$x` and `$y` have been aliased by `foreach`.
- Some patterns including code blocks with syntax errors, such as `/ (?{(^{})/`, would hang

or fail assertions on debugging builds. Now they produce errors.

- An assertion failure when parsing `sort` with debugging enabled has been fixed. *[perl #122771]*.
- `*a = *b; @a = split //, $b[1]` could do a bad read and produce junk results.
- In `() = @array = split`, the `() =` at the beginning no longer confuses the optimizer into assuming a limit of 1.
- Fatal warnings no longer prevent the output of syntax errors. *[perl #122966]*.
- Fixed a NaN double-to-long-double conversion error on VMS. For quiet NaNs (and only on Itanium, not Alpha) negative infinity instead of NaN was produced.
- Fixed the issue that caused `make distclean` to incorrectly leave some files behind. *[perl #122820]*.
- AIX now sets the length in `getsockopt` correctly. *[perl #120835]*. *[cpan #91183]*. *[cpan #85570]*.
- The optimization phase of a regexp compilation could run "forever" and exhaust all memory under certain circumstances; now fixed. *[perl #122283]*.
- The test script *t/op/crypt.t* now uses the SHA-256 algorithm if the default one is disabled, rather than giving failures. *[perl #121591]*.
- Fixed an off-by-one error when setting the size of a shared array. *[perl #122950]*.
- Fixed a bug that could cause perl to enter an infinite loop during compilation. In particular, a `while(1)` within a sublist, *e.g.*

  ```
  sub foo { () = ($a, my $b, ($c, do { while(1) {} })) }
  ```

  The bug was introduced in 5.20.0 *[perl #122995]*.
- On Win32, if a variable was `local`-ized in a pseudo-process that later forked, restoring the original value in the child pseudo-process caused memory corruption and a crash in the child pseudo-process (and therefore the OS process). *[perl #40565]*.
- Calling `write` on a format with a `^**` field could produce a panic in `sv_chop()` if there were insufficient arguments or if the variable used to fill the field was empty. *[perl #123245]*.
- Non-ASCII lexical sub names now appear without trailing junk when they appear in error messages.
- The `\@` subroutine prototype no longer flattens parenthesized arrays (taking a reference to each element), but takes a reference to the array itself. *[perl #47363]*.
- A block containing nothing except a C-style `for` loop could corrupt the stack, causing lists outside the block to lose elements or have elements overwritten. This could happen with `map { for(...){...} } ...` and with lists containing `do { for(...){...} }`. *[perl #123286]*.
- `scalar()` now propagates lvalue context, so that `for(scalar($#foo)) { ... }` can modify `$#foo` through `$_`.
- `qr/@array(?{block})/` no longer dies with "Bizarre copy of ARRAY". *[perl #123344]*.
- `eval '$variable'` in nested named subroutines would sometimes look up a global variable even with a lexical variable in scope.
- In perl 5.20.0, `sort CORE::fake` where 'fake' is anything other than a keyword, started

chopping off the last 6 characters and treating the result as a sort sub name. The previous behavior of treating `CORE::fake` as a sort sub name has been restored. *[perl #123410]*.

- Outside of `use utf8`, a single-character Latin-1 lexical variable is disallowed. The error message for it, "Can't use global `$foo`...", was giving garbage instead of the variable name.
- `readline` on a nonexistent handle was causing `${^LAST_FH}` to produce a reference to an undefined scalar (or fail an assertion). Now `${^LAST_FH}` ends up undefined.
- `(...) x ...` in void context now applies scalar context to the left-hand argument, instead of the context the current sub was called in. *[perl #123020]*.

Known Problems

- `pack`-ing a NaN on a perl compiled with Visual C 6 does not behave properly, leading to a test failure in *t/op/infnan.t*. *[perl 125203]*
- A goal is for Perl to be able to be recompiled to work reasonably well on any Unicode version. In Perl 5.22, though, the earliest such version is Unicode 5.1 (current is 7.0).
- EBCDIC platforms
  - The `cmp` (and hence `sort`) operators do not necessarily give the correct results when both operands are UTF-EBCDIC encoded strings and there is a mixture of ASCII and/or control characters, along with other characters.
  - Ranges containing `\N{...}` in the `tr///` (and `y///`) transliteration operators are treated differently than the equivalent ranges in regular expression patterns. They should, but don't, cause the values in the ranges to all be treated as Unicode code points, and not native ones. (*"Version 8 Regular Expressions" in perlre* gives details as to how it should work.)
  - Encode and encoding are mostly broken.
  - Many CPAN modules that are shipped with core show failing tests.
  - `pack`/`unpack` with `"U0"` format may not work properly.
- The following modules are known to have test failures with this version of Perl. In many cases, patches have been submitted, so there will hopefully be new releases soon:
  - *B::Generate* version 1.50
  - *B::Utils* version 0.25
  - *Coro* version 6.42
  - *Dancer* version 1.3130
  - *Data::Alias* version 1.18
  - *Data::Dump::Streamer* version 2.38
  - *Data::Util* version 0.63
  - *Devel::Spy* version 0.07
  - *invoker* version 0.34
  - *Lexical::Var* version 0.009
  - *LWP::ConsoleLogger* version 0.000018
  - *Mason* version 2.22

- *NgxQueue* version 0.02
- *Padre* version 1.00
- *Parse::Keyword* 0.08

Obituary

Brian McCauley died on May 8, 2015. He was a frequent poster to Usenet, Perl Monks, and other Perl forums, and made several CPAN contributions under the nick NOBULL, including to the Perl FAQ. He attended almost every YAPC::Europe, and indeed, helped organise YAPC::Europe 2006 and the QA Hackathon 2009. His wit and his delight in intricate systems were particularly apparent in his love of board games; many Perl mongers will have fond memories of playing Fluxx and other games with Brian. He will be missed.

Acknowledgements

Perl 5.22.0 represents approximately 12 months of development since Perl 5.20.0 and contains approximately 590,000 lines of changes across 2,400 files from 94 authors.

Excluding auto-generated files, documentation and release tools, there were approximately 370,000 lines of changes to 1,500 .pm, .t, .c and .h files.

Perl continues to flourish into its third decade thanks to a vibrant community of users and developers. The following people are known to have contributed the improvements that became Perl 5.22.0:

Aaron Crane, Abhijit Menon-Sen, Abigail, Alberto SimÃµes, Alex Solovey, Alex Vandiver, Alexandr Ciornii, Alexandre (Midnite) Jousset, Andreas KÃ¶nig, Andreas Voegele, Andrew Fresh, Andy Dougherty, Anthony Heading, Aristotle Pagaltzis, brian d foy, Brian Fraser, Chad Granum, Chris 'BinGOs' Williams, Craig A. Berry, Dagfinn Ilmari MannsÃ¥ker, Daniel Dragan, Darin McBride, Dave Rolsky, David Golden, David Mitchell, David Wheeler, Dmitri Tikhonov, Doug Bell, E. Choroba, Ed J, Eric Herman, Father Chrysostomos, George Greer, Glenn D. Golden, Graham Knop, H.Merijn Brand, Herbert Breunung, Hugo van der Sanden, James E Keenan, James McCoy, James Raspass, Jan Dubois, Jarkko Hietaniemi, Jasmine Ngan, Jerry D. Hedden, Jim Cromie, John Goodyear, kafka, Karen Etheridge, Karl Williamson, Kent Fredric, kmx, Lajos Veres, Leon Timmermans, Lukas Mai, Mathieu Arnold, Matthew Horsfall, Max Maischein, Michael Bunk, Nicholas Clark, Niels Thykier, Niko Tyni, Norman Koch, Olivier MenguÃ©, Peter John Acklam, Peter Martini, Petr PÃ-saÅ™, Philippe Bruhat (BooK), Pierre Bogossian, Rafael Garcia-Suarez, Randy Stauner, Reini Urban, Ricardo Signes, Rob Hoelz, Rostislav Skudnov, Sawyer X, Shirakata Kentaro, Shlomi Fish, Sisyphus, Slaven Rezic, Smylers, Steffen MÃ¼ller, Steve Hay, Sullivan Beck, syber, Tadeusz SoÅ›nierz, Thomas Sibley, Todd Rinaldo, Tony Cook, Vincent Pit, Vladimir Marek, Yaroslav Kuzmin, Yves Orton, Ã†var ArnfjÃ¶rÃ° Bjarmason.

The list above is almost certainly incomplete as it is automatically generated from version control history. In particular, it does not include the names of the (very much appreciated) contributors who reported issues to the Perl bug tracker.

Many of the changes included in this version originated in the CPAN modules included in Perl's core. We're grateful to the entire CPAN community for helping Perl to flourish.

For a more complete list of all of Perl's historical contributors, please see the *AUTHORS* file in the Perl source distribution.

Reporting Bugs

If you find what you think is a bug, you might check the articles recently posted to the comp.lang.perl.misc newsgroup and the perl bug database at *https://rt.perl.org/*. There may also be information at *http://www.perl.org/*, the Perl Home Page.

If you believe you have an unreported bug, please run the *perlbug* program included with your release. Be sure to trim your bug down to a tiny but sufficient test case. Your bug report, along with the output of `perl -V`, will be sent off to perlbug@perl.org to be analysed by the Perl porting team.

If the bug you are reporting has security implications, which make it inappropriate to send to a publicly archived mailing list, then please send it to perl5-security-report@perl.org. This points to a closed subscription unarchived mailing list, which includes all the core committers, who will be able to help assess the impact of issues, figure out a resolution, and help co-ordinate the release of patches to mitigate or fix the problem across all platforms on which Perl is supported. Please only use this address for security issues in the Perl core, not for modules independently distributed on CPAN.

SEE ALSO

The *Changes* file for an explanation of how to view exhaustive details on what changed.

The *INSTALL* file for how to build Perl.

The *README* file for general stuff.

The *Artistic* and *Copying* files for copyright information.