NAME

perl5180delta - what is new for perl v5.18.0

DESCRIPTION

This document describes differences between the v5.16.0 release and the v5.18.0 release.

If you are upgrading from an earlier release such as v5.14.0, first read perl5160delta, which describes differences between v5.14.0 and v5.16.0.

Core Enhancements

New mechanism for experimental features

Newly-added experimental features will now require this incantation:

```
no warnings "experimental::feature_name";
use feature "feature_name"; # would warn without the prev line
```

There is a new warnings category, called ‘‘experimental’’, containing warnings that the feature pragma emits when enabling experimental features.

Newly-added experimental features will also be given special warning IDs, which consist of ‘‘experimental::’’ followed by the name of the feature. (The plan is to extend this mechanism eventually to all warnings, to allow them to be enabled or disabled individually, and not just by category.)

By saying

```
no warnings "experimental::feature_name";
```

you are taking responsibility for any breakage that future changes to, or removal of, the feature may cause.

Since some features (like `~~` or `my $_`) now emit experimental warnings, and you may want to disable them in code that is also run on perls that do not recognize these warning categories, consider using the `if` pragma like this:

```
no if $] >= 5.018, warnings => "experimental::feature_name";
```

Existing experimental features may begin emitting these warnings, too. Please consult perlexperiment(1) for information on which features are considered experimental.

Hash overhaul

Changes to the implementation of hashes in perl v5.18.0 will be one of the most visible changes to the behavior of existing code.

By default, two distinct hash variables with identical keys and values may now provide their contents in a different order where it was previously identical.

When encountering these changes, the key to cleaning up from them is to accept that **hashes are unordered collections** and to act accordingly.

Hash randomization

The seed used by Perl’s hash function is now random. This means that the order which keys/values will be returned from functions like `keys()`, `values()`, and `each()` will differ from run to run.

This change was introduced to make Perl’s hashes more robust to algorithmic complexity attacks, and also because we discovered that it exposes hash ordering dependency bugs and makes them easier to track down.

Toolchain maintainers might want to invest in additional infrastructure to test for things like this. Running tests several times in a row and then comparing results will make it easier to spot hash order dependencies in code. Authors are strongly encouraged not to expose the key order of Perl’s hashes to insecure audiences.

Further, every hash has its own iteration order, which should make it much more difficult to determine what the current hash seed is.

New hash functions

Perl v5.18 includes support for multiple hash functions, and changed the default (to ONE_AT_A_TIME_HARD), you can choose a different algorithm by defining a symbol at compile time. For

a current list, consult the *INSTALL* document. Note that as of Perl v5.18 we can only recommend use of the default or SIPHASH. All the others are known to have security issues and are for research purposes only.

PERL_HASH_SEED environment variable now takes a hex value

`PERL_HASH_SEED` no longer accepts an integer as a parameter; instead the value is expected to be a binary value encoded in a hex string, such as ''0xf5867c55039dc724''. This is to make the infrastructure support hash seeds of arbitrary lengths, which might exceed that of an integer. (SipHash uses a 16 byte seed.)

PERL_PERTURB_KEYS environment variable added

The `PERL_PERTURB_KEYS` environment variable allows one to control the level of randomization applied to `keys` and friends.

When `PERL_PERTURB_KEYS` is 0, perl will not randomize the key order at all. The chance that `keys` changes due to an insert will be the same as in previous perls, basically only when the bucket size is changed.

When `PERL_PERTURB_KEYS` is 1, perl will randomize keys in a non-repeatable way. The chance that `keys` changes due to an insert will be very high. This is the most secure and default mode.

When `PERL_PERTURB_KEYS` is 2, perl will randomize keys in a repeatable way. Repeated runs of the same program should produce the same output every time.

`PERL_HASH_SEED` implies a non-default `PERL_PERTURB_KEYS` setting. Setting `PERL_HASH_SEED=0` (exactly one 0) implies `PERL_PERTURB_KEYS=0` (hash key randomization disabled); setting `PERL_HASH_SEED` to any other value implies `PERL_PERTURB_KEYS=2` (deterministic and repeatable hash key randomization). Specifying `PERL_PERTURB_KEYS` explicitly to a different level overrides this behavior.

Hash::Util::hash_seed() now returns a string

Hash::Util::hash_seed() now returns a string instead of an integer. This is to make the infrastructure support hash seeds of arbitrary lengths which might exceed that of an integer. (SipHash uses a 16 byte seed.)

Output of PERL_HASH_SEED_DEBUG has been changed

The environment variable PERL_HASH_SEED_DEBUG now makes perl show both the hash function perl was built with, *and* the seed, in hex, in use for that process. Code parsing this output, should it exist, must change to accommodate the new format. Example of the new format:

```
$ PERL_HASH_SEED_DEBUG=1 ./perl -e1
HASH_FUNCTION = MURMUR3 HASH_SEED = 0x1476bb9f
```

Upgrade to Unicode 6.2

Perl now supports Unicode 6.2. A list of changes from Unicode 6.1 is at <http://www.unicode.org/versions/Unicode6.2.0>.

Character name aliases may now include non−Latin1−range characters

It is possible to define your own names for characters for use in `\N{...}`, `charnames::vianame()`, etc. These names can now be comprised of characters from the whole Unicode range. This allows for names to be in your native language, and not just English. Certain restrictions apply to the characters that may be used (you can't define a name that has punctuation in it, for example). See ''CUSTOM ALIASES'' in charnames.

New DTrace probes

The following new DTrace probes have been added:

- `op-entry`
- `loading-file`
- `loaded-file`

`${^LAST_FH}`
: This new variable provides access to the filehandle that was last read. This is the handle used by `$.` and by `tell` and `eof` without arguments.

Regular Expression Set Operations
: This is an **experimental** feature to allow matching against the union, intersection, etc., of sets of code points, similar to Unicode::Regex::Set. It can also be used to extend `/x` processing to [bracketed] character classes, and as a replacement of user-defined properties, allowing more complex expressions than they do. See ''Extended Bracketed Character Classes'' in perlrecharclass.

Lexical subroutines
: This new feature is still considered **experimental**. To enable it:

```
use 5.018;
no warnings "experimental::lexical_subs";
use feature "lexical_subs";
```

You can now declare subroutines with `state sub foo`, `my sub foo`, and `our sub foo`. (`state sub` requires that the ''state'' feature be enabled, unless you write it as `CORE::state sub foo`.)

`state sub`creates a subroutine visible within the le xical scope in which it is declared. The subroutine is shared between calls to the outer sub.

`my sub`declares a le xical subroutine that is created each time the enclosing block is entered. `state sub` is generally slightly faster than `my sub`.

`our sub`declares a le xical alias to the package subroutine of the same name.

For more information, see ''Lexical Subroutines'' in perlsub.

Computed Labels
: The loop controls `next`, `last` and `redo`, and the special `dump` operator, now allow arbitrary expressions to be used to compute labels at run time. Previously, any argument that was not a constant was treated as the empty string.

More CORE:: subs
: Several more built-in functions have been added as subroutines to the CORE:: namespace - namely, those non-overridable keywords that can be implemented without custom parsers: `defined`, `delete`, `exists`, `glob`, `pos`, `prototype`, `scalar`, `split`, `study`, and `undef`.

As some of these have prototypes, `prototype('CORE::...')` has been changed to not make a distinction between overridable and non-overridable keywords. This is to make `prototype('CORE::pos')` consistent with `prototype(&CORE::pos)`.

`kill` **with negative signal names**
: `kill` has always allowed a negative signal number, which kills the process group instead of a single process. It has also allowed signal names. But it did not behave consistently, because negative signal names were treated as 0. Now negative signals names like `-INT` are supported and treated the same way as -2 [perl #112990].

Security

See also: hash overhaul
: Some of the changes in the hash overhaul were made to enhance security. Please read that section.

`Storable` **security warning in documentation**
: The documentation for `Storable` now includes a section which warns readers of the danger of accepting Storable documents from untrusted sources. The short version is that deserializing certain types of data can lead to loading modules and other code execution. This is documented behavior and wanted behavior, but this opens an attack vector for malicious entities.

`Locale::Maketext` **allowed code injection via a malicious template**
: If users could provide a translation string to Locale::Maketext, this could be used to invoke arbitrary Perl subroutines available in the current process.

This has been fixed, but it is still possible to invoke any method provided by `Locale::Maketext` itself or a subclass that you are using. One of these methods in turn will invoke the Perl core's `sprintf` subroutine.

In summary, allowing users to provide translation strings without auditing them is a bad idea.

This vulnerability is documented in CVE-2012-6329.

Avoid calling memset with a negative count

Poorly written perl code that allows an attacker to specify the count to perl's `x` string repeat operator can already cause a memory exhaustion denial-of-service attack. A flaw in versions of perl before v5.15.5 can escalate that into a heap buffer overrun; coupled with versions of glibc before 2.16, it possibly allows the execution of arbitrary code.

The flaw addressed to this commit has been assigned identifier CVE-2012-5195 and was researched by Tim Brown.

Incompatible Changes

See also: hash overhaul

Some of the changes in the hash overhaul are not fully compatible with previous versions of perl. Please read that section.

An unknown character name in `\N{...}` is now a syntax error

Previously, it warned, and the Unicode REPLACEMENT CHARACTER was substituted. Unicode now recommends that this situation be a syntax error. Also, the previous behavior led to some confusing warnings and behaviors, and since the REPLACEMENT CHARACTER has no use other than as a stand-in for some unknown character, any code that has this problem is buggy.

Formerly deprecated characters in `\N{}` character name aliases are now errors.

Since v5.12.0, it has been deprecated to use certain characters in user-defined `\N{...}` character names. These now cause a syntax error. For example, it is now an error to begin a name with a digit, such as in

```
my $undraftable = "\N{4F}"; # Syntax error!
```

or to have commas anywhere in the name. See ''CUSTOM ALIASES'' in charnames.

`\N{BELL}` now refers to U+1F514 instead of U+0007

Unicode 6.0 reused the name ''BELL'' for a different code point than it traditionally had meant. Since Perl v5.14, use of this name still referred to U+0007, but would raise a deprecation warning. Now, ''BELL'' refers to U+1F514, and the name for U+0007 is ''ALERT''. All the functions in charnames have been correspondingly updated.

New Restrictions in Multi-Character Case-Insensitive Matching in Regular Expression Bracketed Character Classes

Unicode has now withdrawn their previous recommendation for regular expressions to automatically handle cases where a single character can match multiple characters case-insensitively, for example, the letter LATIN SMALL LETTER SHARP S and the sequence `ss`. This is because it turns out to be impracticable to do this correctly in all circumstances. Because Perl has tried to do this as best it can, it will continue to do so. (We are considering an option to turn it off.) However, a new restriction is being added on such matches when they occur in [bracketed] character classes. People were specifying things such as `/[\0-\xff]/i`, and being surprised that it matches the two character sequence `ss` (since LATIN SMALL LETTER SHARP S occurs in this range). This behavior is also inconsistent with using a property instead of a range: `\p{Block=Latin1}` also includes LATIN SMALL LETTER SHARP S, but `/[\p{Block=Latin1}]/i` does not match `ss`. The new rule is that for there to be a multi-character case-insensitive match within a bracketed character class, the character must be explicitly listed, and not as an end point of a range. This more closely obeys the Principle of Least Astonishment. See ''Bracketed Character Classes'' in perlrecharclass. Note that a bug [perl #89774], now fixed as part of this change, prevented the previous behavior from working fully.

Explicit rules for variable names and identifiers

Due to an oversight, single character variable names in v5.16 were completely unrestricted. This opened the door to several kinds of insanity. As of v5.18, these now follow the rules of other identifiers, in addition to

accepting characters that match the `\p{POSIX_Punct}` property.

There is no longer any difference in the parsing of identifiers specified by using braces versus without braces. For instance, perl used to allow `${foo:bar}` (with a single colon) but not `$foo:bar`. Now that both are handled by a single code path, they are both treated the same way: both are forbidden. Note that this change is about the range of permissible literal identifiers, not other expressions.

Vertical tabs are now whitespace

No one could recall why `\s` didn't match `\cK`, the vertical tab. Now it does. Given the extreme rarity of that character, very little breakage is expected. That said, here's what it means:

`\s` in a regex now matches a vertical tab in all circumstances.

Literal vertical tabs in a regex literal are ignored when the `/x` modifier is used.

Leading vertical tabs, alone or mixed with other whitespace, are now ignored when interpreting a string as a number. For example:

```
$dec = " \cK \t 123";
$hex = " \cK \t 0xF";

say 0 + $dec; # was 0 with warning, now 123
say int $dec; # was 0, now 123
say oct $hex; # was 0, now 15
```

`/(?{})/` and `/(??{})/` have been heavily reworked

The implementation of this feature has been almost completely rewritten. Although its main intent is to fix bugs, some behaviors, especially related to the scope of lexical variables, will have changed. This is described more fully in the ''Selected Bug Fixes'' section.

Stricter parsing of substitution replacement

It is no longer possible to abuse the way the parser parses `s///e` like this:

```
%_=(_,"Just another ");
$_="Perl hacker,\n";
s//_}->{_/e;print
```

`given` now aliases the global `$_`

Instead of assigning to an implicit lexical `$_`, `given` now makes the global `$_` an alias for its argument, just like `foreach`. However, it still uses lexical `$_` if there is lexical `$_` in scope (again, just like `foreach`) [perl #114020].

The smartmatch family of features are now experimental

Smart match, added in v5.10.0 and significantly revised in v5.10.1, has been a regular point of complaint. Although there are a number of ways in which it is useful, it has also proven problematic and confusing for both users and implementors of Perl. There have been a number of proposals on how to best address the problem. It is clear that smartmatch is almost certainly either going to change or go away in the future. Relying on its current behavior is not recommended.

Warnings will now be issued when the parser sees `~~`, `given`, or `when`. To disable these warnings, you can add this line to the appropriate scope:

```
no if $] >= 5.018, warnings => "experimental::smartmatch";
```

Consider, though, replacing the use of these features, as they may change behavior again before becoming stable.

Lexical `$_` is now experimental

Since it was introduced in Perl v5.10, it has caused much confusion with no obvious solution:

- Various modules (e.g., List::Util) expect callback routines to use the global `$_`. `use List::Util 'first'; my $_; first { $_ == 1 } @list` does not w ork as one would expect.
- A `my $_` declaration earlier in the same file can cause confusing closure warnings.

- The ‘‘_’’ subroutine prototype character allows called subroutines to access your lexical $_, so it is not really private after all.
- Nevertheless, subroutines with a ‘‘(@)’’ prototype and methods cannot access the caller’s lexical $_, unless they are written in XS.
- But even XS routines cannot access a lexical $_ declared, not in the calling subroutine, but in an outer scope, iff that subroutine happened not to mention $_ or use any operators that default to $_.

It is our hope that lexical $_ can be rehabilitated, but this may cause changes in its behavior. Please use it with caution until it becomes stable.

readline() **with** `$/ = \N` **now reads N characters, not N bytes**

Previously, when reading from a stream with I/O layers such as `encoding`, the *readline()* function, otherwise known as the <> operator, would read *N* bytes from the top-most layer. [perl #79960]

Now, *N* characters are read instead.

There is no change in behaviour when reading from streams with no extra layers, since bytes map exactly to characters.

Overridden `glob` **is now passed one argument**

`glob` overrides used to be passed a magical undocumented second argument that identified the caller. Nothing on CPAN was using this, and it got in the way of a bug fix, so it was removed. If you really need to identify the caller, see Devel::Callsite on CPAN.

Here doc parsing

The body of a here document inside a quote-like operator now always begins on the line after the ‘‘<<foo’’ marker. Previously, it was documented to begin on the line following the containing quote-like operator, but that was only sometimes the case [perl #114040].

Alphanumeric operators must now be separated from the closing delimiter of regular expressions

You may no longer write something like:

```
m/a/and 1
```

Instead you must write

```
m/a/ and 1
```

with whitespace separating the operator from the closing delimiter of the regular expression. Not having whitespace has resulted in a deprecation warning since Perl v5.14.0.

qw(...) can no longer be used as parentheses

`qw` lists used to fool the parser into thinking they were always surrounded by parentheses. This permitted some surprising constructions such as `foreach $x qw(a b c) {...}`, which should really be written `foreach $x (qw(a b c)) {...}`. These would sometimes get the lexer into the wrong state, so they didn’t fully work, and the similar `foreach qw(a b c) {...}` that one might expect to be permitted never worked at all.

This side effect of `qw` has now been abolished. It has been deprecated since Perl v5.13.11. It is now necessary to use real parentheses everywhere that the grammar calls for them.

Interaction of lexical and default warnings

Turning on any lexical warnings used first to disable all default warnings if lexical warnings were not already enabled:

```
$*; # deprecation warning
use warnings "void";
$#; # void warning; no deprecation warning
```

Now, the `debugging`, `deprecated`, `glob`, `inplace` and `malloc` warnings categories are left on when turning on lexical warnings (unless they are turned off by `no warnings`, of course).

This may cause deprecation warnings to occur in code that used to be free of warnings.

Those are the only categories consisting only of default warnings. Default warnings in other categories are still disabled by `use warnings "category"`, as we do not yet have the infrastructure for controlling individual warnings.

`state sub` **and** `our sub`

Due to an accident of history, `state sub` and `our sub` were equivalent to a plain `sub`, so one could even create an anonymous sub with `our sub { ... }`. These are now disallowed outside of the ''lexical_subs'' feature. Under the ''lexical_subs'' feature they have new meanings described in ''Lexical Subroutines'' in perlsub.

Defined values stored in environment are forced to byte strings

A value stored in an environment variable has always been stringified when inherited by child processes.

In this release, when assigning to `%ENV`, values are immediately stringified, and converted to be only a byte string.

First, it is forced to be a only a string. Then if the string is utf8 and the equivalent of `utf8::downgrade()` works, that result is used; otherwise, the equivalent of `utf8::encode()` is used, and a warning is issued about wide characters (''Diagnostics'').

`require` **dies for unreadable files**

When `require` encounters an unreadable file, it now dies. It used to ignore the file and continue searching the directories in `@INC` [perl #113422].

`gv_fetchmeth_*` **and SUPER**

The various `gv_fetchmeth_*` XS functions used to treat a package whose named ended with `::SUPER` specially. A method lookup on the `Foo::SUPER` package would be treated as a `SUPER` method lookup on the `Foo` package. This is no longer the case. To do a `SUPER` lookup, pass the `Foo` stash and the `GV_SUPER` flag.

`split`**'s first argument is more consistently interpreted**

After some changes earlier in v5.17, `split`'s behavior has been simplified: if the PATTERN argument evaluates to a string containing one space, it is treated the way that a *literal* string containing one space once was.

Deprecations

Module removals

The following modules will be removed from the core distribution in a future release, and will at that time need to be installed from CPAN. Distributions on CPAN which require these modules will need to list them as prerequisites.

The core versions of these modules will now issue `"deprecated"`-category warnings to alert you to this fact. To silence these deprecation warnings, install the modules in question from CPAN.

Note that these are (with rare exceptions) fine modules that you are encouraged to continue to use. Their disinclusion from core primarily hinges on their necessity to bootstrapping a fully functional, CPAN-capable Perl installation, not usually on concerns over their design.

encoding

The use of this pragma is now strongly discouraged. It conflates the encoding of source text with the encoding of I/O data, reinterprets escape sequences in source text (a questionable choice), and introduces the UTF-8 bug to all runtime handling of character strings. It is broken as designed and beyond repair.

For using non-ASCII literal characters in source text, please refer to utf8. For dealing with textual I/O data, please refer to Encode and open.

Archive::Extract
B::Lint
B::Lint::Debug

CPANPLUS and all included `CPANPLUS::*` modules
Devel::InnerPackage
Log::Message
Log::Message::Config
Log::Message::Handlers
Log::Message::Item
Log::Message::Simple
Module::Pluggable
Module::Pluggable::Object
Object::Accessor
Pod::LaTeX
Term::UI
Term::UI::History

Deprecated Utilities

The following utilities will be removed from the core distribution in a future release as their associated modules have been deprecated. They will remain available with the applicable CPAN distribution.

cpanp
`cpanp-run-perl`
cpan2dist
These items are part of the `CPANPLUS` distribution.

pod2latex
This item is part of the `Pod::LaTeX` distribution.

PL_sv_objcount

This interpreter-global variable used to track the total number of Perl objects in the interpreter. It is no longer maintained and will be removed altogether in Perl v5.20.

Five additional characters should be escaped in patterns with `/x`

When a regular expression pattern is compiled with `/x`, Perl treats 6 characters as white space to ignore, such as SPACE and TAB. However, Unicode recommends 11 characters be treated thusly. We will conform with this in a future Perl version. In the meantime, use of any of the missing characters will raise a deprecation warning, unless turned off. The five characters 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
```

User-defined charnames with surprising whitespace

A user-defined character name with trailing or multiple spaces in a row is likely a typo. This now generates a warning when defined, on the assumption that uses of it will be unlikely to include the excess whitespace.

Various XS-callable functions are now deprecated

All the functions used to classify characters will be removed from a future version of Perl, and should not be used. With participating C compilers (e.g., gcc), compiling any file that uses any of these will generate a warning. These were not intended for public use; there are equivalent, faster, macros for most of them.

See ‘‘Character classes’’ in perlapi. The complete list is:

`is_uni_alnum`, `is_uni_alnumc`, `is_uni_alnumc_lc`, `is_uni_alnum_lc`, `is_uni_alpha`, `is_uni_alpha_lc`, `is_uni_ascii`, `is_uni_ascii_lc`, `is_uni_blank`, `is_uni_blank_lc`, `is_uni_cntrl`, `is_uni_cntrl_lc`, `is_uni_digit`, `is_uni_digit_lc`, `is_uni_graph`, `is_uni_graph_lc`, `is_uni_idfirst`, `is_uni_idfirst_lc`, `is_uni_lower`, `is_uni_lower_lc`, `is_uni_print`, `is_uni_print_lc`, `is_uni_punct`, `is_uni_punct_lc`, `is_uni_space`, `is_uni_space_lc`, `is_uni_upper`, `is_uni_upper_lc`, `is_uni_xdigit`,

`is_uni_xdigit_lc`, `is_utf8_alnum`, `is_utf8_alnumc`, `is_utf8_alpha`, `is_utf8_ascii`, `is_utf8_blank`, `is_utf8_char`, `is_utf8_cntrl`, `is_utf8_digit`, `is_utf8_graph`, `is_utf8_idcont`, `is_utf8_idfirst`, `is_utf8_lower`, `is_utf8_mark`, `is_utf8_perl_space`, `is_utf8_perl_word`, `is_utf8_posix_digit`, `is_utf8_print`, `is_utf8_punct`, `is_utf8_space`, `is_utf8_upper`, `is_utf8_xdigit`, `is_utf8_xidcont`, `is_utf8_xidfirst`.

In addition these three functions that have never worked properly are deprecated: `to_uni_lower_lc`, `to_uni_title_lc`, and `to_uni_upper_lc`.

Certain rare uses of backslashes within regexes are now deprecated

There are three pairs of characters that Perl recognizes as metacharacters in regular expression patterns: `{}`, `[]`, and `()`. These can be used as well to delimit patterns, as in:

```
m{foo}
s(foo)(bar)
```

Since they are metacharacters, they have special meaning to regular expression patterns, and it turns out that you can't turn off that special meaning by the normal means of preceding them with a backslash, if you use them, paired, within a pattern delimited by them. For example, in

```
m{foo\{1,3\}}
```

the backslashes do not change the behavior, and this matches `"fo"` followed by one to three more occurrences of `"o"`.

Usages like this, where they are interpreted as metacharacters, are exceedingly rare; we think there are none, for example, in all of CPAN. Hence, this deprecation should affect very little code. It does give notice, however, that any such code needs to change, which will in turn allow us to change the behavior in future Perl versions so that the backslashes do have an effect, and without fear that we are silently breaking any existing code.

Splitting the tokens `(?` and `(*` in regular expressions

A deprecation warning is now raised if the `(` and `?` are separated by white space or comments in `(?...)` regular expression constructs. Similarly, if the `(` and `*` are separated in `(*VERB...)` constructs.

Pre-PerlIO IO implementations

In theory, you can currently build perl without PerlIO. Instead, you'd use a wrapper around stdio or sfio. In practice, this isn't very useful. It's not well tested, and without any support for IO layers or (thus) Unicode, it's not much of a perl. Building without PerlIO will most likely be removed in the next version of perl.

PerlIO supports a `stdio` layer if stdio use is desired. Similarly a sfio layer could be produced in the future, if needed.

Future Deprecations

- Platforms without support infrastructure

  Both Windows CE and z/OS have been historically under-maintained, and are currently neither successfully building nor regularly being smoke tested. Efforts are underway to change this situation, but it should not be taken for granted that the platforms are safe and supported. If they do not become buildable and regularly smoked, support for them may be actively removed in future releases. If you have an interest in these platforms and you can lend your time, expertise, or hardware to help support these platforms, please let the perl development effort know by emailing `perl5-porters@perl.org`.

  Some platforms that appear otherwise entirely dead are also on the short list for removal between now and v5.20.0:

  DG/UX
  NeXT

  We also think it likely that current versions of Perl will no longer build AmigaOS, DJGPP, NetWare (natively), OS/2 and Plan 9. If you are using Perl on such a platform and have an interest in ensuring

Perl's future on them, please contact us.

We believe that Perl has long been unable to build on mixed endian architectures (such as PDP-11s), and intend to remove any remaining support code. Similarly, code supporting the long umaintained GNU dld will be removed soon if no-one makes themselves known as an active user.

- Swapping of $< and $>

  Perl has supported the idiom of swapping $< and $> (and likewise $(and $)) to temporarily drop permissions since 5.0, like this:

  ```
  ($<, $>) = ($>, $<);
  ```

  However, this idiom modifies the real user/group id, which can have undesirable side-effects, is no longer useful on any platform perl supports and complicates the implementation of these variables and list assignment in general.

  As an alternative, assignment only to `$>` is recommended:

  ```
  local $> = $<;
  ```

  See also: Setuid Demystified <http://www.cs.berkeley.edu/~daw/papers/setuid-usenix02.pdf>.

- `microperl`, long broken and of unclear present purpose, will be removed.
- Revamping `"\Q"` semantics in double-quotish strings when combined with other escapes.

  There are several bugs and inconsistencies involving combinations of `\Q` and escapes like `\x`, `\L`, etc., within a `\Q...\E` pair. These need to be fixed, and doing so will necessarily change current behavior. The changes have not yet been settled.

- Use of `$x`, where `x` stands for any actual (non-printing) C0 control character will be disallowed in a future Perl version. Use `${x}` instead (where again `x` stands for a control character), or better, `$^A`, where `^` is a caret (CIRCUMFLEX ACCENT), and `A` stands for any of the characters listed at the end of "OPERATOR DIFFERENCES" in perlebcdic.

Performance Enhancements

- Lists of lexical variable declarations (`my($x, $y)`) are now optimised down to a single op and are hence faster than before.
- A new C preprocessor define `NO_TAINT_SUPPORT` was added that, if set, disables Perl's taint support altogether. Using the -T or -t command line flags will cause a fatal error. Beware that both core tests as well as many a CPAN distribution's tests will fail with this change. On the upside, it provides a small performance benefit due to reduced branching.

  Do not enable this unless you know exactly what you are getting yourself into.

- `pack` with constant arguments is now constant folded in most cases [perl #113470].
- Speed up in regular expression matching against Unicode properties. The largest gain is for `\X`, the Unicode "extended grapheme cluster." The gain for it is about 35% - 40%. Bracketed character classes, e.g., `[0-9\x{100}]` containing code points above 255 are also now faster.
- On platforms supporting it, several former macros are now implemented as static inline functions. This should speed things up slightly on non-GCC platforms.
- The optimisation of hashes in boolean context has been extended to affect `scalar(%hash)`, `%hash ? ... : ...`, and `sub { %hash || ... }`.
- Filetest operators manage the stack in a fractionally more efficient manner.
- Globs used in a numeric context are now numified directly in most cases, rather than being numified via stringification.
- The `x` repetition operator is now folded to a single constant at compile time if called in scalar context with constant operands and no parentheses around the left operand.

Modules and Pragmata

New Modules and Pragmata

- Config::Perl::V version 0.16 has been added as a dual-lifed module. It provides structured data retrieval of `perl -V` output including information only known to the `perl` binary and not available via Config.

Updated Modules and Pragmata

For a complete list of updates, run:

```
$ corelist --diff 5.16.0 5.18.0
```

You can substitute your favorite version in place of `5.16.0`, too.

- Archive::Extract has been upgraded to 0.68.

  Work around an edge case on Linux with Busybox's unzip.

- Archive::Tar has been upgraded to 1.90.

  ptar now supports the -T option as well as dashless options [rt.cpan.org #75473], [rt.cpan.org #75475].

  Auto-encode filenames marked as UTF-8 [rt.cpan.org #75474].

  Don't use `tell` on IO::Zlib handles [rt.cpan.org #64339].

  Don't try to `chown` on symlinks.

- autodie has been upgraded to 2.13.

  `autodie` now plays nicely with the 'open' pragma.

- B has been upgraded to 1.42.

  The `stashoff` method of COPs has been added. This provides access to an internal field added in perl 5.16 under threaded builds [perl #113034].

  `B::COP::stashpv` now supports UTF-8 package names and embedded NULs.

  All `CVf_*` and `GVf_*` and more SV-related flag values are now provided as constants in the `B::` namespace and available for export. The default export list has not changed.

  This makes the module work with the new pad API.

- B::Concise has been upgraded to 0.95.

  The `-nobanner` option has been fixed, and `format`s can now be dumped. When passed a sub name to dump, it will check also to see whether it is the name of a format. If a sub and a format share the same name, it will dump both.

  This adds support for the new `OpMAYBE_TRUEBOOL` and `OPpTRUEBOOL` flags.

- B::Debug has been upgraded to 1.18.

  This adds support (experimentally) for `B::PADLIST` which was added in Perl 5.17.4.

- B::Deparse has been upgraded to 1.20.

  Avoid warning when run under `perl -w`.

  It now deparses loop controls with the correct precedence, and multiple statements in a `format` line are also now deparsed correctly.

  This release suppresses trailing semicolons in formats.

  This release adds stub deparsing for lexical subroutines.

  It no longer dies when deparsing `sort` without arguments. It now correctly omits the comma for `system $prog @args`and `exec $prog @args`.

- bignum, bigint and bigrat have been upgraded to 0.33.

  The overrides for `hex` and `oct` have been rewritten, eliminating several problems, and making one incompatible change:

  - Formerly, whichever of `use bigint` or `use bigrat` was compiled later would take precedence over the other, causing `hex` and `oct` not to respect the other pragma when in scope.
  - Using any of these three pragmata would cause `hex` and `oct` anywhere else in the program to evaluate their arguments in list context and prevent them from inferring `$_` when called without arguments.
  - Using any of these three pragmata would make `oct("1234")` return 1234 (for any number not beginning with 0) anywhere in the program. Now ‘‘1234’’ is translated from octal to decimal, whether within the pragma’s scope or not.
  - The global overrides that facilitate lexical use of `hex` and `oct` now respect any existing overrides that were in place before the new overrides were installed, falling back to them outside of the scope of `use bignum`.
  - `use bignum "hex"`, `use bignum "oct"` and similar invocations for bigint and bigrat now export a `hex` or `oct` function, instead of providing a global override.

- Carp has been upgraded to 1.29.

  Carp is no longer confused when `caller` returns undef for a package that has been deleted.

  The `longmess()` and `shortmess()` functions are now documented.

- CGI has been upgraded to 3.63.

  Unrecognized HTML escape sequences are now handled better, problematic trailing newlines are no longer inserted after <form> tags by `startform()` or `start_form()`, and bogus ‘‘Insecure Dependency’’ warnings appearing with some versions of perl are now worked around.

- Class::Struct has been upgraded to 0.64.

  The constructor now respects overridden accessor methods [perl #29230].

- Compress::Raw::Bzip2 has been upgraded to 2.060.

  The misuse of Perl’s ‘‘magic’’ API has been fixed.

- Compress::Raw::Zlib has been upgraded to 2.060.

  Upgrade bundled zlib to version 1.2.7.

  Fix build failures on Irix, Solaris, and Win32, and also when building as C++ [rt.cpan.org #69985], [rt.cpan.org #77030], [rt.cpan.org #75222].

  The misuse of Perl’s ‘‘magic’’ API has been fixed.

  `compress()`, `uncompress()`, `memGzip()` and `memGunzip()` have been speeded up by making parameter validation more efficient.

- CPAN::Meta::Requirements has been upgraded to 2.122.

  Treat undef requirements to `from_string_hash` as 0 (with a warning).

  Added `requirements_for_module` method.

- CPANPLUS has been upgraded to 0.9135.

  Allow adding *blib/script* to PATH.

  Save the history between invocations of the shell.

  Handle multiple `makemakerargs` and `makeflags` arguments better.

  This resolves issues with the SQLite source engine.

- Data::Dumper has been upgraded to 2.145.

  It has been optimized to only build a seen-scalar hash as necessary, thereby speeding up serialization drastically.

  Additional tests were added in order to improve statement, branch, condition and subroutine coverage. On the basis of the coverage analysis, some of the internals of Dumper.pm were refactored. Almost all methods are now documented.

- DB_File has been upgraded to 1.827.

  The main Perl module no longer uses the `"@_"` construct.

- Devel::Peek has been upgraded to 1.11.

  This fixes compilation with C++ compilers and makes the module work with the new pad API.

- Digest::MD5 has been upgraded to 2.52.

  Fix `Digest::Perl::MD5` OO fallback [rt.cpan.org #66634].

- Digest::SHA has been upgraded to 5.84.

  This fixes a double-free bug, which might have caused vulnerabilities in some cases.

- DynaLoader has been upgraded to 1.18.

  This is due to a minor code change in the XS for the VMS implementation.

  This fixes warnings about using `CODE` sections without an `OUTPUT` section.

- Encode has been upgraded to 2.49.

  The Mac alias x-mac-ce has been added, and various bugs have been fixed in Encode::Unicode, Encode::UTF7 and Encode::GSM0338.

- Env has been upgraded to 1.04.

  Its SPLICE implementation no longer misbehaves in list context.

- ExtUtils::CBuilder has been upgraded to 0.280210.

  Manifest files are now correctly embedded for those versions of VC++ which make use of them. [perl #111782, #111798].

  A list of symbols to export can now be passed to `link()` when on Windows, as on other OSes [perl #115100].

- ExtUtils::ParseXS has been upgraded to 3.18.

  The generated C code now avoids unnecessarily incrementing `PL_amagic_generation` on Perl versions where it's done automatically (or on current Perl where the variable no longer exists).

  This avoids a bogus warning for initialised XSUB non-parameters [perl #112776].

- File::Copy has been upgraded to 2.26.

  `copy()` no longer zeros files when copying into the same directory, and also now fails (as it has long been documented to do) when attempting to copy a file over itself.

- File::DosGlob has been upgraded to 1.10.

  The internal cache of file names that it keeps for each caller is now freed when that caller is freed. This means `use File::DosGlob 'glob'; eval 'scalar <*>'` no longer leaks memory .

- File::Fetch has been upgraded to 0.38.

  Added the 'file_default' option for URLs that do not have a file component.

  Use `File::HomeDir` when available, and provide `PERL5_CPANPLUS_HOME` to override the autodetection.

Always re-fetch *CHECKSUMS* if `fetchdir` is set.

- File::Find has been upgraded to 1.23.

  This fixes inconsistent unixy path handling on VMS.

  Individual files may now appear in list of directories to be searched [perl #59750].

- File::Glob has been upgraded to 1.20.

  File::Glob has had exactly the same fix as File::DosGlob. Since it is what Perl's own `glob` operator itself uses (except on VMS), this means `eval 'scalar <*>'` no longer leaks.

  A space-separated list of patterns return long lists of results no longer results in memory corruption or crashes. This bug was introduced in Perl 5.16.0. [perl #114984]

- File::Spec::Unix has been upgraded to 3.40.

  `abs2rel` could produce incorrect results when given two relative paths or the root directory twice [perl #111510].

- File::stat has been upgraded to 1.07.

  `File::stat` ignores the filetest pragma, and warns when used in combination therewith. But it was not warning for `-r`. This has been fixed [perl #111640].

  `-p` now works, and does not return false for pipes [perl #111638].

  Previously `File::stat` overloaded `-x` and `-X` operators did not give the correct results for directories or executable files when running as root. They had been treating executable permissions for root just like for any other user, performing group membership tests *etc* for files not owned by root. They now follow the correct Unix behaviour - for a directory they are always true, and for a file if any of the three execute permission bits are set then they report that root can execute the file. Perl's builtin `-x` and `-X` operators have always been correct.

- File::Temp has been upgraded to 0.23

  Fixes various bugs involving directory removal. Defers unlinking tempfiles if the initial unlink fails, which fixes problems on NFS.

- GDBM_File has been upgraded to 1.15.

  The undocumented optional fifth parameter to `TIEHASH` has been removed. This was intended to provide control of the callback used by `gdbm*` functions in case of fatal errors (such as filesystem problems), but did not work (and could never have worked). No code on CPAN even attempted to use it. The callback is now always the previous default, `croak`. Problems on some platforms with how the `C croak` function is called have also been resolved.

- Hash::Util has been upgraded to 0.15.

  `hash_unlocked` and `hashref_unlocked` now returns true if the hash is unlocked, instead of always returning false [perl #112126].

  `hash_unlocked`, `hashref_unlocked`, `lock_hash_recurse` and `unlock_hash_recurse` are now exportable [perl #112126].

  Two new functions, `hash_locked` and `hashref_locked`, have been added. Oddly enough, these two functions were already exported, even though they did not exist [perl #112126].

- HTTP::Tiny has been upgraded to 0.025.

  Add SSL verification features [github #6], [github #9].

  Include the final URL in the response hashref.

  Add `local_address` option.

  This improves SSL support.

- IO has been upgraded to 1.28.

  `sync()` can now be called on read-only file handles [perl #64772].

  IO::Socket tries harder to cache or otherwise fetch socket information.

- IPC::Cmd has been upgraded to 0.80.

  Use `POSIX::_exit` instead of `exit` in `run_forked` [rt.cpan.org #76901].

- IPC::Open3 has been upgraded to 1.13.

  The `open3()` function no longer uses `POSIX::close()` to close file descriptors since that breaks the ref-counting of file descriptors done by PerlIO in cases where the file descriptors are shared by PerlIO streams, leading to attempts to close the file descriptors a second time when any such PerlIO streams are closed later on.

- Locale::Codes has been upgraded to 3.25.

  It includes some new codes.

- Memoize has been upgraded to 1.03.

  Fix the `MERGE` cache option.

- Module::Build has been upgraded to 0.4003.

  Fixed bug where modules without `$VERSION` might have a version of '0' listed in 'provides' metadata, which will be rejected by PAUSE.

  Fixed bug in PodParser to allow numerals in module names.

  Fixed bug where giving arguments twice led to them becoming arrays, resulting in install paths like *ARRAY(0xdeadbeef)/lib/Foo.pm*.

  A minor bug fix allows markup to be used around the leading "Name" in a POD "abstract" line, and some documentation improvements have been made.

- Module::CoreList has been upgraded to 2.90

  Version information is now stored as a delta, which greatly reduces the size of the *CoreList.pm* file.

  This restores compatibility with older versions of perl and cleans up the corelist data for various modules.

- Module::Load::Conditional has been upgraded to 0.54.

  Fix use of `requires` on perls installed to a path with spaces.

  Various enhancements include the new use of Module::Metadata.

- Module::Metadata has been upgraded to 1.000011.

  The creation of a Module::Metadata object for a typical module file has been sped up by about 40%, and some spurious warnings about `$VERSION`s have been suppressed.

- Module::Pluggable has been upgraded to 4.7.

  Amongst other changes, triggers are now allowed on events, which gives a powerful way to modify behaviour.

- Net::Ping has been upgraded to 2.41.

  This fixes some test failures on Windows.

- Opcode has been upgraded to 1.25.

  Reflect the removal of the boolkeys opcode and the addition of the clonecv, introcv and padcv opcodes.

- overload has been upgraded to 1.22.

  `no overload`no w warns for invalid arguments, just like `use overload`.

- PerlIO::encoding has been upgraded to 0.16.

  This is the module implementing the ‘‘:encoding(...)’’ I/O layer. It no longer corrupts memory or crashes when the encoding back-end reallocates the buffer or gives it a typeglob or shared hash key scalar.

- PerlIO::scalar has been upgraded to 0.16.

  The buffer scalar supplied may now only contain code points 0xFF or lower. [perl #109828]

- Perl::OSType has been upgraded to 1.003.

  This fixes a bug detecting the VOS operating system.

- Pod::Html has been upgraded to 1.18.

  The option `--libpods` has been reinstated. It is deprecated, and its use does nothing other than issue a warning that it is no longer supported.

  Since the HTML files generated by pod2html claim to have a UTF-8 charset, actually write the files out using UTF-8 [perl #111446].

- Pod::Simple has been upgraded to 3.28.

  Numerous improvements have been made, mostly to Pod::Simple::XHTML, which also has a compatibility change: the `codes_in_verbatim` option is now disabled by default. See *cpan/Pod-Simple/ChangeLog* for the full details.

- re has been upgraded to 0.23

  Single character [class]es like `/[s]/` or `/[s]/i` are now optimized as if they did not have the brackets, i.e. `/s/` or `/s/i`.

  See note about `op_comp` in the ‘‘Internal Changes’’ section below.

- Safe has been upgraded to 2.35.

  Fix interactions with `Devel::Cover`

  Don’t eval code under `no strict`.

- Scalar::Util has been upgraded to version 1.27.

  Fix an overloading issue with `sum`.

  `first` and `reduce` now check the callback first (so `&first(1)` is disallowed).

  Fix `tainted` on magical values [rt.cpan.org #55763].

  Fix `sum` on previously magical values [rt.cpan.org #61118].

  Fix reading past the end of a fixed buffer [rt.cpan.org #72700].

- Search::Dict has been upgraded to 1.07.

  No longer require `stat` on filehandles.

  Use `fc` for casefolding.

- Socket has been upgraded to 2.009.

  Constants and functions required for IP multicast source group membership have been added.

  `unpack_sockaddr_in()` and `unpack_sockaddr_in6()` now return just the IP address in scalar context, and `inet_ntop()` now guards against incorrect length scalars being passed in.

  This fixes an uninitialized memory read.

- Storable has been upgraded to 2.41.

  Modifying `$_[0]` within `STORABLE_freeze` no longer results in crashes [perl #112358].

  An object whose class implements `STORABLE_attach` is now thawed only once when there are multiple references to it in the structure being thawed [perl #111918].

  Restricted hashes were not always thawed correctly [perl #73972].

  Storable would croak when freezing a blessed REF object with a `STORABLE_freeze()` method [perl #113880].

  It can now freeze and thaw vstrings correctly. This causes a slight incompatible change in the storage format, so the format version has increased to 2.9.

  This contains various bugfixes, including compatibility fixes for older versions of Perl and vstring handling.

- Sys::Syslog has been upgraded to 0.32.

  This contains several bug fixes relating to `getservbyname()`, `setlogsock()`and log levels in `syslog()`, together with fixes for Windows, Haiku-OS and GNU/kFreeBSD. See *cpan/Sys-Syslog/Changes* for the full details.

- Term::ANSIColor has been upgraded to 4.02.

  Add support for italics.

  Improve error handling.

- Term::ReadLine has been upgraded to 1.10. This fixes the use of the **cpan** and **cpanp** shells on Windows in the event that the current drive happens to contain a *\dev\tty* file.

- Test::Harness has been upgraded to 3.26.

  Fix glob semantics on Win32 [rt.cpan.org #49732].

  Don't use `Win32::GetShortPathName` when calling perl [rt.cpan.org #47890].

  Ignore -T when reading shebang [rt.cpan.org #64404].

  Handle the case where we don't know the wait status of the test more gracefully.

  Make the test summary 'ok' line overridable so that it can be changed to a plugin to make the output of prove idempotent.

  Don't run world-writable files.

- Text::Tabs and Text::Wrap have been upgraded to 2012.0818. Support for Unicode combining characters has been added to them both.

- threads::shared has been upgraded to 1.31.

  This adds the option to warn about or ignore attempts to clone structures that can't be cloned, as opposed to just unconditionally dying in that case.

  This adds support for dual-valued values as created by Scalar::Util::dualvar.

- Tie::StdHandle has been upgraded to 4.3.

  `READ` now respects the offset argument to `read` [perl #112826].

- Time::Local has been upgraded to 1.2300.

  Seconds values greater than 59 but less than 60 no longer cause `timegm()` and `timelocal()` to croak.

- Unicode::UCD has been upgraded to 0.53.

  This adds a function *all_casefolds()* that returns all the casefolds.

- Win32 has been upgraded to 0.47.

  New APIs have been added for getting and setting the current code page.

Removed Modules and Pragmata

- Version::Requirements has been removed from the core distribution. It is available under a different name: CPAN::Meta::Requirements.

Documentation

Changes to Existing Documentation

perlcheat(1)

- perlcheat(1) has been reorganized, and a few new sections were added.

perldata(1)

- Now explicitly documents the behaviour of hash initializer lists that contain duplicate keys.

perldiag(1)

- The explanation of symbolic references being prevented by ‘‘strict refs’’ now doesn’t assume that the reader knows what symbolic references are.

perlfaq(1)

- perlfaq(1) has been synchronized with version 5.0150040 from CPAN.

perlfunc(1)

- The return value of `pipe` is now documented.
- Clarified documentation of `our`.

perlop(1)

- Loop control verbs (`dump`, `goto`, `next`, `last` and `redo`) have always had the same precedence as assignment operators, but this was not documented until now.

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

- Unterminated delimiter for here document

  This message now occurs when a here document label has an initial quotation mark but the final quotation mark is missing.

  This replaces a bogus and misleading error message about not finding the label itself [perl #114104].

- panic: child pseudo-process was never scheduled

  This error is thrown when a child pseudo-process in the ithreads implementation on Windows was not scheduled within the time period allowed and therefore was not able to initialize properly [perl #88840].

- Group name must start with a non-digit word character in regex; marked by <-- HERE in m/%s/

  This error has been added for `(?&0)`, which is invalid. It used to produce an incomprehensible error message [perl #101666].

- Can’t use an undefined value as a subroutine reference

  Calling an undefined value as a subroutine now produces this error message. It used to, but was accidentally disabled, first in Perl 5.004 for non-magical variables, and then in Perl v5.14 for magical (e.g., tied) variables. It has now been restored. In the mean time, undef was treated as an empty string [perl #113576].

- Experimental ‘‘%s’’ subs not enabled

  To use lexical subs, you must first enable them:

  ```
  no warnings 'experimental::lexical_subs';
  use feature 'lexical_subs';
  my sub foo { ... }
  ```

New Warnings

- ’Strings with code points over 0xFF may not be mapped into in-memory file handles’
- ’%s’ resolved to ’\o{%s}%d’
- ’Trailing white-space in a charnames alias definition is deprecated’
- ’A sequence of multiple spaces in a charnames alias definition is deprecated’
- ’Passing malformed UTF-8 to ‘‘%s’’ is deprecated’
- Subroutine ‘‘&%s’’ is not available

  (W closure) During compilation, an inner named subroutine or eval is attempting to capture an outer lexical subroutine that is not currently available. This can happen for one of two reasons. First, the lexical subroutine may be declared in an outer anonymous subroutine that has not yet been created. (Remember that named subs are created at compile time, while anonymous subs are created at run-time.) For example,

  ```
  sub { my sub a {...} sub f { \&a } }
  ```

  At the time that f is created, it can’t capture the current the ‘‘a’’ sub, since the anonymous subroutine hasn’t been created yet. Conversely, the following won’t give a warning since the anonymous subroutine has by now been created and is live:

  ```
  sub { my sub a {...} eval 'sub f { \&a }' }->();
  ```

  The second situation is caused by an eval accessing a variable that has gone out of scope, for example,

  ```
  sub f {
  my sub a {...}
  sub { eval '\&a' }
  }
  f()->();
  ```

  Here, when the ’\&a’ in the eval is being compiled, f() is not currently being executed, so its &a is not available for capture.

- ‘‘%s’’ subroutine &%s masks earlier declaration in same `%s`

  (W misc) A ‘‘my’’ or ‘‘state’’ subroutine has been redeclared in the current scope or statement, effectively eliminating all access to the previous instance. This is almost always a typographical error. Note that the earlier subroutine will still exist until the end of the scope or until all closure references to it are destroyed.

- The `%s` feature is experimental

  (S experimental) This warning is emitted if you enable an experimental feature via `use feature`. 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::lexical_subs";
  use feature "lexical_subs";
  ```

- sleep(%u) too large

  (W overflow) You called `sleep` with a number that was larger than it can reliably handle and `sleep` probably slept for less time than requested.

- Wide character in setenv

  Attempts to put wide characters into environment variables via `%ENV` now provoke this warning.

- "Invalid negative number (%s) in chr"

  `chr()` now warns when passed a negative value [perl #83048].

- "Integer overflow in srand"

  `srand()` now warns when passed a value that doesn't fit in a `UV` (since the value will be truncated rather than overflowing) [perl #40605].

- "-i used with no filenames on the command line, reading from STDIN"

  Running perl with the `-i` flag now warns if no input files are provided on the command line [perl #113410].

Changes to Existing Diagnostics

- $* is no longer supported

  The warning that use of `$*` and `$#` is no longer supported is now generated for every location that references them. Previously it would fail to be generated if another variable using the same typeglob was seen first (e.g. `@*` before `$*`), and would not be generated for the second and subsequent uses. (It's hard to fix the failure to generate warnings at all without also generating them every time, and warning every time is consistent with the warnings that `$[` used to generate.)

- The warnings for `\b{` and `\B{` were added. They are a deprecation warning which should be turned off by that category. One should not have to turn off regular regexp warnings as well to get rid of these.

- Constant(%s): Call to &{$^H{%s}} did not return a defined value

  Constant overloading that returns `undef` results in this error message. For numeric constants, it used to say ''Constant(undef)''. ''undef'' has been replaced with the number itself.

- The error produced when a module cannot be loaded now includes a hint that the module may need to be installed: ''Can't locate hopping.pm in `@INC` (you may need to install the hopping module) (@INC contains: ...)''

- vector argument not supported with alpha versions

  This warning was not suppressible, even with `no warnings`. Now it is suppressible, and has been moved from the ''internal'' category to the ''printf'' category.

- `Can't do {n,m} with n > m in regex; marked by <-- HERE in m/%s/`

  This fatal error has been turned into a warning that reads:

  Quantifier {n,m} with n > m can't match in regex

  (W regexp) Minima should be less than or equal to maxima. If you really want your regexp to match something 0 times, just put {0}.

- The ''Runaway prototype'' warning that occurs in bizarre cases has been removed as being unhelpful and inconsistent.

- The ''Not a format reference'' error has been removed, as the only case in which it could be triggered was a bug.

- The ''Unable to create sub named `%s`'' error has been removed for the same reason.

- The 'Can't use ''my `%s`'' in sort comparison' error has been downgraded to a warning, ''''my `%s`'' used in sort comparison' (with 'state' instead of 'my' for state variables). In addition, the heuristics for guessing whether lexical `$a` or `$b` has been misused have been improved to generate fewer false positives. Lexical `$a` and `$b` are no longer disallowed if they are outside the sort block. Also, a named unary or list operator inside the sort block no longer causes the `$a` or `$b` to be ignored [perl #86136].

Utility Changes

h2xs

- *h2xs* no longer produces invalid code for empty defines. [perl #20636]

Configuration and Compilation

- Added `useversionedarchname` option to Configure

  When set, it includes 'api_versionstring' in 'archname'. E.g. x86_64-linux-5.13.6-thread-multi. It is unset by default.

  This feature was requested by Tim Bunce, who observed that `INSTALL_BASE` creates a library structure that does not differentiate by perl version. Instead, it places architecture specific files in ''$install_base/lib/perl5/$archname''. This makes it difficult to use a common `INSTALL_BASE` library path with multiple versions of perl.

  By setting `-Duseversionedarchname`, the `$archname` will be distinct for architecture *and* API version, allowing mixed use of `INSTALL_BASE`.

- Add a `PERL_NO_INLINE_FUNCTIONS` option

  If `PERL_NO_INLINE_FUNCTIONS` is defined, don't include ''inline.h''

  This permits test code to include the perl headers for definitions without creating a link dependency on the perl library (which may not exist yet).

- Configure will honour the external `MAILDOMAIN` environment variable, if set.
- `installman` no longer ignores the silent option
- Both `META.yml` and `META.json` files are now included in the distribution.
- *Configure* will now correctly detect `isblank()` when compiling with a C++ compiler.
- The pager detection in *Configure* has been improved to allow responses which specify options after the program name, e.g. **/usr/bin/less -R**, if the user accepts the default value. This helps **perldoc(1)** when handling ANSI escapes [perl #72156].

Testing

- The test suite now has a section for tests that require very large amounts of memory. These tests won't run by default; they can be enabled by setting the `PERL_TEST_MEMORY` environment variable to the number of gibibytes of memory that may be safely used.

Platform Support

Discontinued Platforms

BeOS

BeOS was an operating system for personal computers developed by Be Inc, initially for their BeBox hardware. The OS Haiku was written as an open source replacement for/continuation of BeOS, and its perl port is current and actively maintained.

UTS Global

Support code relating to UTS global has been removed. UTS was a mainframe version of System V created by Amdahl, subsequently sold to UTS Global. The port has not been touched since before Perl v5.8.0, and UTS Global is now defunct.

VM/ESA

Support for VM/ESA has been removed. The port was tested on 2.3.0, which IBM ended service on in March 2002. 2.4.0 ended service in June 2003, and was superseded by Z/VM. The current version of Z/VM is V6.2.0, and scheduled for end of service on 2015/04/30.

MPE/IX

Support for MPE/IX has been removed.

EPOC
: Support code relating to EPOC has been removed. EPOC was a family of operating systems developed by Psion for mobile devices. It was the predecessor of Symbian. The port was last updated in April 2002.

Rhapsody
: Support for Rhapsody has been removed.

Platform-Specific Notes

AIX

Configure now always adds `-qlanglvl=extc99` to the CC flags on AIX when using xlC. This will make it easier to compile a number of XS-based modules that assume C99 [perl #113778].

clang++

There is now a workaround for a compiler bug that prevented compiling with clang++ since Perl v5.15.7 [perl #112786].

C++

When compiling the Perl core as C++ (which is only semi-supported), the mathom functions are now compiled as `extern "C"`, to ensure proper binary compatibility. (However, binary compatibility isn't generally guaranteed anyway in the situations where this would matter.)

Darwin

Stop hardcoding an alignment on 8 byte boundaries to fix builds using -Dusemorebits.

Haiku

Perl should now work out of the box on Haiku R1 Alpha 4.

MidnightBSD

`libc_r` was removed from recent versions of MidnightBSD and older versions work better with `pthread`. Threading is now enabled using `pthread` which corrects build errors with threading enabled on 0.4-CURRENT.

Solaris

In Configure, avoid running sed commands with flags not supported on Solaris.

VMS

- Where possible, the case of filenames and command-line arguments is now preserved by enabling the CRTL features `DECC$EFS_CASE_PRESERVE` and `DECC$ARGV_PARSE_STYLE` at start-up time. The latter only takes effect when extended parse is enabled in the process from which Perl is run.
- The character set for Extended Filename Syntax (EFS) is now enabled by default on VMS. Among other things, this provides better handling of dots in directory names, multiple dots in filenames, and spaces in filenames. To obtain the old behavior, set the logical name `DECC$EFS_CHARSET` to `DISABLE`.
- Fixed linking on builds configured with `-Dusemymalloc=y`.
- Experimental support for building Perl with the HP C++ compiler is available by configuring with `-Dusecxx`.
- All C header files from the top-level directory of the distribution are now installed on VMS, providing consistency with a long-standing practice on other platforms. Previously only a subset were installed, which broke non-core extension builds for extensions that depended on the missing include files.
- Quotes are now removed from the command verb (but not the parameters) for commands spawned via `system`, backticks, or a piped `open`. Previously, quotes on the verb were passed through to DCL, which would fail to recognize the command. Also, if the verb is actually a path to an image or command procedure on an ODS-5 volume, quoting it now allows the path to contain spaces.

- The **a2p** build has been fixed for the HP C++ compiler on OpenVMS.

Win32

- Perl can now be built using Microsoft's Visual C++ 2012 compiler by specifying CCTYPE=MSVC110 (or MSVC110FREE if you are using the free Express edition for Windows Desktop) in *win32/Makefile*.
- The option to build without `USE_SOCKETS_AS_HANDLES` has been removed.
- Fixed a problem where perl could crash while cleaning up threads (including the main thread) in threaded debugging builds on Win32 and possibly other platforms [perl #114496].
- A rare race condition that would lead to sleep taking more time than requested, and possibly even hanging, has been fixed [perl #33096].
- `link` on Win32 now attempts to set `$!` to more appropriate values based on the Win32 API error code. [perl #112272]

  Perl no longer mangles the environment block, e.g. when launching a new sub-process, when the environment contains non-ASCII characters. Known problems still remain, however, when the environment contains characters outside of the current ANSI codepage (e.g. see the item about Unicode in `%ENV` in <http://perl5.git.perl.org/perl.git/blob/HEAD:/Porting/todo.pod>). [perl #113536]
- Building perl with some Windows compilers used to fail due to a problem with miniperl's `glob` operator (which uses the `perlglob` program) deleting the PATH environment variable [perl #113798].
- A new makefile option, `USE_64_BIT_INT`, has been added to the Windows makefiles. Set this to "define" when building a 32-bit perl if you want it to use 64-bit integers.

  Machine code size reductions, already made to the DLLs of XS modules in Perl v5.17.2, have now been extended to the perl DLL itself.

  Building with VC++ 6.0 was inadvertently broken in Perl v5.17.2 but has now been fixed again.

WinCE

Building on WinCE is now possible once again, although more work is required to fully restore a clean build.

Internal Changes

- Synonyms for the misleadingly named `av_len()` have been created: `av_top_index()` and `av_tindex`. All three of these return the number of the highest index in the array, not the number of elements it contains.
- *SvUPGRADE()* is no longer an expression. Originally this macro (and its underlying function, *sv_upgrade()*) were documented as boolean, although in reality they always croaked on error and never returned false. In 2005 the documentation was updated to specify a void return value, but *SvUPGRADE()* was left always returning 1 for backwards compatibility. This has now been removed, and *SvUPGRADE()* is now a statement with no return value.

  So this is now a syntax error:

  ```
  if (!SvUPGRADE(sv)) { croak(...); }
  ```

  If you have code like that, simply replace it with

  ```
  SvUPGRADE(sv);
  ```

  or to avoid compiler warnings with older perls, possibly

  ```
  (void)SvUPGRADE(sv);
  ```
- Perl has a new copy-on-write mechanism that allows any SvPOK scalar to be upgraded to a copy-on-write scalar. A reference count on the string buffer is stored in the string buffer itself. This feature is **not enabled by default**.

It can be enabled in a perl build by running *Configure* with **-Accflags=-DPERL_NEW_COPY_ON_WRITE**, and we would encourage XS authors to try their code with such an enabled perl, and provide feedback. Unfortunately, there is not yet a good guide to updating XS code to cope with COW. Until such a document is available, consult the perl5-porters mailing list.

It breaks a few XS modules by allowing copy-on-write scalars to go through code paths that never encountered them before.

- Copy-on-write no longer uses the SvFAKE and SvREADONLY flags. Hence, SvREADONLY indicates a true read-only SV.

  Use the SvIsCOW macro (as before) to identify a copy-on-write scalar.

- `PL_glob_index` is gone.
- The private Perl_croak_no_modify has had its context parameter removed. It is now has a void prototype. Users of the public API croak_no_modify remain unaffected.
- Copy-on-write (shared hash key) scalars are no longer marked read-only. `SvREADONLY` returns false on such an SV, but `SvIsCOW` still returns true.
- A new op type, `OP_PADRANGE` has been introduced. The perl peephole optimiser will, where possible, substitute a single padrange op for a pushmark followed by one or more pad ops, and possibly also skipping list and nextstate ops. In addition, the op can carry out the tasks associated with the RHS of a `my(...) = @_` assignment, so those ops may be optimised away too.
- Case-insensitive matching inside a [bracketed] character class with a multi-character fold no longer excludes one of the possibilities in the circumstances that it used to. [perl #89774].
- `PL_formfeed` has been removed.
- The regular expression engine no longer reads one byte past the end of the target string. While for all internally well-formed scalars this should never have been a problem, this change facilitates clever tricks with string buffers in CPAN modules. [perl #73542]
- Inside a BEGIN block, `PL_compcv` now points to the currently-compiling subroutine, rather than the BEGIN block itself.
- `mg_length` has been deprecated.
- `sv_len` now always returns a byte count and `sv_len_utf8` a character count. Previously, `sv_len` and `sv_len_utf8` were both buggy and would sometimes returns bytes and sometimes characters. `sv_len_utf8` no longer assumes that its argument is in UTF-8. Neither of these creates UTF-8 caches for tied or overloaded values or for non-PVs any more.
- `sv_mortalcopy` now copies string buffers of shared hash key scalars when called from XS modules [perl #79824].
- The new `RXf_MODIFIES_VARS` flag can be set by custom regular expression engines to indicate that the execution of the regular expression may cause variables to be modified. This lets `s///` know to skip certain optimisations. Perl's own regular expression engine sets this flag for the special backtracking verbs that set `$REGMARK` and `$REGERROR`.
- The APIs for accessing lexical pads have changed considerably.

  `PADLIST`s are now longer AVs, but their own type instead. `PADLIST`s no w contain a `PAD` and a `PADNAMELIST` of `PADNAME`s, rather than AVs for the pad and the list of pad names. `PAD`s, `PADNAMELIST`s, and `PADNAME`s are to be accessed as such through the newly added pad API instead of the plain AV and SV APIs. See perlapi(1) for details.

- In the regex API, the numbered capture callbacks are passed an index indicating what match variable is being accessed. There are special index values for the `$``, `$&`, `$&` variables. Previously the same three values were used to retrieve `${^PREMATCH}`, `${^MATCH}`, `${^POSTMATCH}` too, but these have now been assigned three separate values. See ''Numbered capture callbacks'' in perlreapi.

- `PL_sawampersand` was previously a boolean indicating that any of `$``, `$&`, `$'` had been seen; it now contains three one-bit flags indicating the presence of each of the variables individually.
- The `CV *` typemap entry now supports `&{}` overloading and typeglobs, just like `&{...}` [perl #96872].
- The `SVf_AMAGIC` flag to indicate overloading is now on the stash, not the object. It is now set automatically whenever a method or `@ISA` changes, so its meaning has changed, too. It now means ''potentially overloaded''. When the overload table is calculated, the flag is automatically turned off if there is no overloading, so there should be no noticeable slowdown.

  The staleness of the overload tables is now checked when overload methods are invoked, rather than during `bless`.

  ''A'' magic is gone. The changes to the handling of the `SVf_AMAGIC` flag eliminate the need for it.

  `PL_amagic_generation` has been removed as no longer necessary. For XS modules, it is now a macro alias to `PL_na`.

  The fallback overload setting is now stored in a stash entry separate from overloadedness itself.
- The character-processing code has been cleaned up in places. The changes should be operationally invisible.
- The `study` function was made a no-op in v5.16. It was simply disabled via a `return` statement; the code was left in place. Now the code supporting what `study` used to do has been removed.
- Under threaded perls, there is no longer a separate PV allocated for every COP to store its package name (`cop->stashpv`). Instead, there is an offset (`cop->stashoff`) into the new `PL_stashpad` array, which holds stash pointers.
- In the pluggable regex API, the `regexp_engine` struct has acquired a new field `op_comp`, which is currently just for perl's internal use, and should be initialized to NULL by other regex plugin modules.
- A new function `alloccopstash` has been added to the API, but is considered experimental. See perlapi.
- Perl used to implement get magic in a way that would sometimes hide bugs in code that could call *mg_get()* too many times on magical values. This hiding of errors no longer occurs, so long-standing bugs may become visible now. If you see magic-related errors in XS code, check to make sure it, together with the Perl API functions it uses, calls *mg_get()* only once on *SvGMAGICAL()* values.
- OP allocation for CVs now uses a slab allocator. This simplifies memory management for OPs allocated to a CV, so cleaning up after a compilation error is simpler and safer [perl #111462][perl #112312].
- `PERL_DEBUG_READONLY_OPS` has been rewritten to work with the new slab allocator, allowing it to catch more violations than before.
- The old slab allocator for ops, which was only enabled for `PERL_IMPLICIT_SYS` and `PERL_DEBUG_READONLY_OPS`, has been retired.

Selected Bug Fixes

- Here document terminators no longer require a terminating newline character when they occur at the end of a file. This was already the case at the end of a string eval [perl #65838].
- `-DPERL_GLOBAL_STRUCT` builds now free the global struct **after** they've finished using it.
- A trailing '/' on a path in `@INC` will no longer have an additional '/' appended.
- The `:crlf` layer now works when unread data doesn't fit into its own buffer. [perl #112244].
- `ungetc()` now handles UTF-8 encoded data. [perl #116322].
- A bug in the core typemap caused any C types that map to the T_BOOL core typemap entry to not be set, updated, or modified when the T_BOOL variable was used in an OUTPUT: section with an exception for RETVAL. T_BOOL in an INPUT: section was not affected. Using a T_BOOL return type

for an XSUB (RETVAL) was not affected. A side effect of fixing this bug is, if a T_BOOL is specified in the OUTPUT: section (which previous did nothing to the SV), and a read only SV (literal) is passed to the XSUB, croaks like ‘‘Modification of a read-only value attempted’’ will happen. [perl #115796]

- On many platforms, providing a directory name as the script name caused perl to do nothing and report success. It should now universally report an error and exit nonzero. [perl #61362]
- `sort {undef} ...`under f atal warnings no longer crashes. It had begun crashing in Perl v5.16.
- Stashes blessed into each other (`bless \%Foo::, 'Bar'; bless \%Bar::, 'Foo'`) no longer result in double frees. This bug started happening in Perl v5.16.
- Numerous memory leaks have been fixed, mostly involving fatal warnings and syntax errors.
- Some failed regular expression matches such as `'f' =~ /../g` were not resetting `pos`. Also, ‘‘match-once’’ patterns (`m?...?g`) failed to reset it, too, when invoked a second time [perl #23180].
- Several bugs involving `local *ISA` and `local *Foo::` causing stale MRO caches have been fixed.
- Defining a subroutine when its typeglob has been aliased no longer results in stale method caches. This bug was introduced in Perl v5.10.
- Localising a typeglob containing a subroutine when the typeglob’s package has been deleted from its parent stash no longer produces an error. This bug was introduced in Perl v5.14.
- Under some circumstances, `local *method=...` would fail to reset method caches upon scope exit.
- `/[.foo.]/` is no longer an error, but produces a warning (as before) and is treated as `/[.fo]/` [perl #115818].
- `goto $tied_var`no w calls FETCH before deciding what type of goto (subroutine or label) this is.
- Renaming packages through glob assignment (`*Foo:: = *Bar::; *Bar:: = *Baz::`) in combination with `m?...?` and `reset` no longer makes threaded builds crash.
- A number of bugs related to assigning a list to hash have been fixed. Many of these involve lists with repeated keys like `(1, 1, 1, 1)`.
  - The expression `scalar(%h = (1, 1, 1, 1))` now returns `4`, not `2`.
  - The return value of `%h = (1, 1, 1)` in list context was wrong. Previously this would return `(1, undef, 1)`, now it returns `(1, undef)`.
  - Perl now issues the same warning on `($s, %h) = (1, {})` as it does for `(%h) = ({})`, ‘‘Reference found where even-sized list expected’’.
  - A number of additional edge cases in list assignment to hashes were corrected. For more details see commit 23b7025ebc.
- Attributes applied to lexical variables no longer leak memory. [perl #114764]
- `dump`, `goto`, `last`, `next`, `redo` or `require` followed by a bareword (or version) and then an infix operator is no longer a syntax error. It used to be for those infix operators (like `+`) that have a different meaning where a term is expected. [perl #105924]
- `require a::b . 1`and `require a::b + 1` no longer produce erroneous ambiguity warnings. [perl #107002]
- Class method calls are now allowed on any string, and not just strings beginning with an alphanumeric character. [perl #105922]
- An empty pattern created with `qr//` used in `m///` no longer triggers the ‘‘empty pattern reuses last pattern’’ behaviour. [perl #96230]
- Tying a hash during iteration no longer results in a memory leak.

- Freeing a tied hash during iteration no longer results in a memory leak.
- List assignment to a tied array or hash that dies on STORE no longer results in a memory leak.
- If the hint hash (`%^H`) is tied, compile-time scope entry (which copies the hint hash) no longer leaks memory if FETCH dies. [perl #107000]
- Constant folding no longer inappropriately triggers the special `split " "` behaviour. [perl #94490]
- `defined scalar(@array)`, `defined do { &foo }`, and similar constructs now treat the argument to `defined` as a simple scalar. [perl #97466]
- Running a custom debugging that defines no `*DB::DB` glob or provides a subroutine stub for `&DB::DB` no longer results in a crash, but an error instead. [perl #114990]
- `reset ""`no w matches its documentation. `reset` only resets `m?...?` patterns when called with no argument. An empty string for an argument now does nothing. (It used to be treated as no argument.) [perl #97958]
- `printf` with an argument returning an empty list no longer reads past the end of the stack, resulting in erratic behaviour. [perl #77094]
- `--subname` no longer produces erroneous ambiguity warnings. [perl #77240]
- `v10` is now allowed as a label or package name. This was inadvertently broken when v-strings were added in Perl v5.6. [perl #56880]
- `length`, `pos`, `substr` and `sprintf` could be confused by ties, overloading, references and typeglobs if the stringification of such changed the internal representation to or from UTF-8. [perl #114410]
- utf8::encode now calls FETCH and STORE on tied variables. utf8::decode now calls STORE (it was already calling FETCH).
- `$tied =~ s/$non_utf8/$utf8/`no longer loops infinitely if the tied v ariable returns a Latin-1 string, shared hash key scalar, or reference or typeglob that stringifies as ASCII or Latin-1. This was a regression from v5.12.
- `s///` without /e is now better at detecting when it needs to forego certain optimisations, fixing some buggy cases:
  - Match variables in certain constructs (`&&`, `||`, `..` and others) in the replacement part; e.g., `s/(.)/$l{$a||$1}/g`. [perl #26986]
  - Aliases to match variables in the replacement.
  - `$REGERROR` or `$REGMARK` in the replacement. [perl #49190]
  - An empty pattern (`s//$foo/`) that causes the last-successful pattern to be used, when that pattern contains code blocks that modify the variables in the replacement.
- The taintedness of the replacement string no longer affects the taintedness of the return value of `s///e`.
- The `$|` autoflush variable is created on-the-fly when needed. If this happened (e.g., if it was mentioned in a module or eval) when the currently-selected filehandle was a typeglob with an empty IO slot, it used to crash. [perl #115206]
- Line numbers at the end of a string eval are no longer off by one. [perl #114658]
- `@INC` filters (subroutines returned by subroutines in `@INC`) that set `$_` to a copy-on-write scalar no longer cause the parser to modify that string buffer in place.
- `length($object)` no longer returns the undefined value if the object has string overloading that returns undef. [perl #115260]
- The use of `PL_stashcache`, the stash name lookup cache for method calls, has been restored,

  Commit da6b625f78f5f133 in August 2011 inadvertently broke the code that looks up values in

`PL_stashcache`. As it's a only cache, quite correctly everything carried on working without it.

- The error ''Can't localize through a reference'' had disappeared in v5.16.0 when `local %$ref` appeared on the last line of an lvalue subroutine. This error disappeared for `\local %$ref` in perl v5.8.1. It has now been restored.
- The parsing of here-docs has been improved significantly, fixing several parsing bugs and crashes and one memory leak, and correcting wrong subsequent line numbers under certain conditions.
- Inside an eval, the error message for an unterminated here-doc no longer has a newline in the middle of it [perl #70836].
- A substitution inside a substitution pattern (`s/${s|||}//`) no longer confuses the parser.
- It may be an odd place to allow comments, but `s//"" # hello/e` has always worked, *unless* there happens to be a null character before the first #. Now it works even in the presence of nulls.
- An invalid range in `tr///` or `y///` no longer results in a memory leak.
- String eval no longer treats a semicolon-delimited quote-like operator at the very end (`eval 'q;;'`) as a syntax error.
- `warn {$_ => 1} + 1`is no longer a syntax error . The parser used to get confused with certain list operators followed by an anonymous hash and then an infix operator that shares its form with a unary operator.
- `(caller $n)[6]`(which gi ves the text of the eval) used to return the actual parser buffer. Modifying it could result in crashes. Now it always returns a copy. The string returned no longer has ''\n;'' tacked on to the end. The returned text also includes here-doc bodies, which used to be omitted.
- The UTF-8 position cache is now reset when accessing magical variables, to avoid the string buffer and the UTF-8 position cache getting out of sync [perl #114410].
- Various cases of get magic being called twice for magical UTF-8 strings have been fixed.
- This code (when not in the presence of $& etc)

  ```
  $_ = 'x' x 1_000_000;
  1 while /(.)/;
  ```

  used to skip the buffer copy for performance reasons, but suffered from `$1` etc changing if the original string changed. That's now been fixed.
- Perl doesn't use PerlIO anymore to report out of memory messages, as PerlIO might attempt to allocate more memory.
- In a regular expression, if something is quantified with `{n,m}` where `n>m`, it can't possibly match. Previously this was a fatal error, but now is merely a warning (and that something won't match). [perl #82954].
- It used to be possible for formats defined in subroutines that have subsequently been undefined and redefined to close over variables in the wrong pad (the newly-defined enclosing sub), resulting in crashes or ''Bizarre copy'' errors.
- Redefinition of XSUBs at run time could produce warnings with the wrong line number.
- The `%vd` sprintf format does not support version objects for alpha versions. It used to output the format itself (%vd) when passed an alpha version, and also emit an ''Invalid conversion in printf'' warning. It no longer does, but produces the empty string in the output. It also no longer leaks memory in this case.
- `$obj->SUPER::method` calls in the main package could fail if the SUPER package had already been accessed by other means.
- Stash aliasing (`*foo:: = *bar::`) no longer causes SUPER calls to ignore changes to methods or `@ISA` or use the wrong package.

- Method calls on packages whose names end in ::SUPER are no longer treated as SUPER method calls, resulting in failure to find the method. Furthermore, defining subroutines in such packages no longer causes them to be found by SUPER method calls on the containing package [perl #114924].
- `\w` now matches the code points U+200C (ZERO WIDTH NON-JOINER) and U+200D (ZERO WIDTH JOINER). `\W` no longer matches these. This change is because Unicode corrected their definition of what `\w` should match.
- `dump LABEL`no longer leaks its label.
- Constant folding no longer changes the behaviour of functions like `stat()` and `truncate()` that can take either filenames or handles. `stat 1 ? foo : bar` no ws treats its argument as a file name (since it is an arbitrary expression), rather than the handle ‘‘foo’’.
- `truncate FOO, $len`no longer f alls back to treating ‘‘FOO’’ as a file name if the filehandle has been deleted. This was broken in Perl v5.16.0.
- Subroutine redefinitions after sub-to-glob and glob-to-glob assignments no longer cause double frees or panic messages.
- `s///` now turns vstrings into plain strings when performing a substitution, even if the resulting string is the same (`s/a/a/`).
- Prototype mismatch warnings no longer erroneously treat constant subs as having no prototype when they actually have "".
- Constant subroutines and forward declarations no longer prevent prototype mismatch warnings from omitting the sub name.
- `undef` on a subroutine now clears call checkers.
- The `ref` operator started leaking memory on blessed objects in Perl v5.16.0. This has been fixed [perl #114340].
- `use` no longer tries to parse its arguments as a statement, making `use constant { () };` a syntax error [perl #114222].
- On debugging builds, ‘‘uninitialized’’ warnings inside formats no longer cause assertion failures.
- On debugging builds, subroutines nested inside formats no longer cause assertion failures [perl #78550].
- Formats and `use` statements are now permitted inside formats.
- `print $x`and `sub { print $x }->()` no w always produce the same output. It was possible for the latter to refuse to close over `$x` if the variable was not active; e.g., if it was defined outside a currently-running named subroutine.
- Similarly, `print $x` and `print eval '$x'` now produce the same output. This also allows ‘‘my `$x` if 0’’ variables to be seen in the debugger [perl #114018].
- Formats called recursively no longer stomp on their own lexical variables, but each recursive call has its own set of lexicals.
- Attempting to free an active format or the handle associated with it no longer results in a crash.
- Format parsing no longer gets confused by braces, semicolons and low-precedence operators. It used to be possible to use braces as format delimiters (instead of `=` and `.`), but only sometimes. Semicolons and low-precedence operators in format argument lines no longer confuse the parser into ignoring the line’s return value. In format argument lines, braces can now be used for anonymous hashes, instead of being treated always as `do` blocks.
- Formats can now be nested inside code blocks in regular expressions and other quoted constructs (`/(?{...})/` and `qq/${...}/`) [perl #114040].
- Formats are no longer created after compilation errors.

- Under debugging builds, the **-DA** command line option started crashing in Perl v5.16.0. It has been fixed [perl #114368].
- A potential deadlock scenario involving the premature termination of a pseudo- forked child in a Windows build with ithreads enabled has been fixed. This resolves the common problem of the *t/op/fork.t* test hanging on Windows [perl #88840].
- The code which generates errors from `require()` could potentially read one or two bytes before the start of the filename for filenames less than three bytes long and ending `/\.p?\z/`. This has now been fixed. Note that it could never have happened with module names given to `use()` or `require()` anyway.
- The handling of pathnames of modules given to `require()` has been made thread-safe on VMS.
- Non-blocking sockets have been fixed on VMS.
- Pod can now be nested in code inside a quoted construct outside of a string eval. This used to work only within string evals [perl #114040].
- `goto ''` no w looks for an empty label, producing the ‘‘goto must have label’’ error message, instead of exiting the program [perl #111794].
- `goto "\0"` no w dies with ‘‘Can’t find label’’ instead of ‘‘goto must have label’’.
- The C function `hv_store` used to result in crashes when used on `%^H` [perl #111000].
- A call checker attached to a closure prototype via `cv_set_call_checker` is now copied to closures cloned from it. So `cv_set_call_checker` now works inside an attribute handler for a closure.
- Writing to `$^N` used to have no effect. Now it croaks with ‘‘Modification of a read-only value’’ by default, but that can be overridden by a custom regular expression engine, as with `$1` [perl #112184].
- `undef` on a control character glob (`undef *^H`) no longer emits an erroneous warning about ambiguity [perl #112456].
- For efficiency’s sake, many operators and built-in functions return the same scalar each time. Lvalue subroutines and subroutines in the CORE:: namespace were allowing this implementation detail to leak through. `print &CORE::uc("a"), &CORE::uc("b")` used to print ‘‘BB’’. The same thing would happen with an lvalue subroutine returning the return value of `uc`. Now the value is copied in such cases.
- `method {}` syntax with an empty block or a block returning an empty list used to crash or use some random value left on the stack as its invocant. Now it produces an error.
- `vec` now works with extremely large offsets (>2 GB) [perl #111730].
- Changes to overload settings now take effect immediately, as do changes to inheritance that affect overloading. They used to take effect only after `bless`.

  Objects that were created before a class had any overloading used to remain non-overloaded even if the class gained overloading through `use overload` or `@ISA` changes, and even after `bless`. This has been fixed [perl #112708].
- Classes with overloading can now inherit fallback values.
- Overloading was not respecting a fallback value of 0 if there were overloaded objects on both sides of an assignment operator like `+=` [perl #111856].
- `pos` now croaks with hash and array arguments, instead of producing erroneous warnings.
- `while(each %h)` no w implies `while(defined($_ = each %h))`, like `readline` and `readdir`.
- Subs in the CORE:: namespace no longer crash after `undef *_` when called with no argument list (`&CORE::time` with no parentheses).

- `unpack` no longer produces the "'/' must follow a numeric type in unpack" error when it is the data that are at fault [perl #60204].
- `join` and `"@array"` now call FETCH only once on a tied `$"` [perl #8931].
- Some subroutine calls generated by compiling core ops affected by a `CORE::GLOBAL` override had op checking performed twice. The checking is always idempotent for pure Perl code, but the double checking can matter when custom call checkers are involved.
- A race condition used to exist around fork that could cause a signal sent to the parent to be handled by both parent and child. Signals are now blocked briefly around fork to prevent this from happening [perl #82580].
- The implementation of code blocks in regular expressions, such as `(?{})` and `(??{})`, has been heavily reworked to eliminate a whole slew of bugs. The main user-visible changes are:
  - Code blocks within patterns are now parsed in the same pass as the surrounding code; in particular it is no longer necessary to have balanced braces: this now works:

    ```
    /(?{ $x='{' })/
    ```

    This means that this error message is no longer generated:

    ```
    Sequence (?{...}) not terminated or not {}-balanced in regex
    ```

    but a new error may be seen:

    ```
    Sequence (?{...}) not terminated with ')'
    ```

    In addition, literal code blocks within run-time patterns are only compiled once, at perl compile-time:

    ```
    for my $p (...) {
    # this 'FOO' block of code is compiled once,
    # at the same time as the surrounding 'for' loop
    /$p{(?{FOO;})/;
    }
    ```
  - Lexical variables are now sane as regards scope, recursion and closure behavior. In particular, `/A(?{B})C/` behaves (from a closure viewpoint) exactly like `/A/ && do { B } && /C/`, while `qr/A(?{B})C/` is like `sub {/A/ && do { B } && /C/}`. So this code now works how you might expect, creating three regexes that match 0, 1, and 2:

    ```
    for my $i (0..2) {
    push @r, qr/^(??{$i})$/;
    }
    "1" =~ $r[1]; # matches
    ```
  - The `use re 'eval'` pragma is now only required for code blocks defined at runtime; in particular in the following, the text of the `$r` pattern is still interpolated into the new pattern and recompiled, but the individual compiled code-blocks within `$r` are reused rather than being recompiled, and `use re 'eval'` isn't needed any more:

    ```
    my $r = qr/abc(?{....})def/;
    /xyz$r/;
    ```
  - Flow control operators no longer crash. Each code block runs in a new dynamic scope, so `next` etc. will not see any enclosing loops. `return` returns a value from the code block, not from any enclosing subroutine.
  - Perl normally caches the compilation of run-time patterns, and doesn't recompile if the pattern hasn't changed, but this is now disabled if required for the correct behavior of closures. For example:

```
my $code = '(??{$x})';
for my $x (1..3) {
# recompile to see fresh value of $x each time
$x =~ /$code/;
}
```

- The `/msix` and `(?msix)` etc. flags are now propagated into the return value from `(??{})`; this now works:

  ```
  "AB" =~ /a(??{'b'})/i;
  ```

- Warnings and errors will appear to come from the surrounding code (or for run-time code blocks, from an eval) rather than from an `re_eval`:

  ```
  use re 'eval'; $c = '(?{ warn "foo" })'; /$c/;
  /(?{ warn "foo" })/;
  ```

  formerly gave:

  ```
  foo at (re_eval 1) line 1.
  foo at (re_eval 2) line 1.
  ```

  and now gives:

  ```
  foo at (eval 1) line 1.
  foo at /some/prog line 2.
  ```

- Perl now can be recompiled to use any Unicode version. In v5.16, it worked on Unicodes 6.0 and 6.1, but there were various bugs if earlier releases were used; the older the release the more problems.
- `vec` no longer produces ‘‘uninitialized’’ warnings in lvalue context [perl #9423].
- An optimization involving fixed strings in regular expressions could cause a severe performance penalty in edge cases. This has been fixed [perl #76546].
- In certain cases, including empty subpatterns within a regular expression (such as `(?:)` or `(?:|)`) could disable some optimizations. This has been fixed.
- The ‘‘Can’t find an opnumber’’ message that `prototype` produces when passed a string like ‘‘CORE::nonexistent_keyword’’ now passes UTF-8 and embedded NULs through unchanged [perl #97478].
- `prototype` now treats magical variables like `$1` the same way as non-magical variables when checking for the CORE:: prefix, instead of treating them as subroutine names.
- Under threaded perls, a runtime code block in a regular expression could corrupt the package name stored in the op tree, resulting in bad reads in `caller`, and possibly crashes [perl #113060].
- Referencing a closure prototype (`\&{$_[1]}` in an attribute handler for a closure) no longer results in a copy of the subroutine (or assertion failures on debugging builds).
- `eval '__PACKAGE__'` no w returns the right answer on threaded builds if the current package has been assigned over (as in `*ThisPackage:: = *ThatPackage::`) [perl #78742].
- If a package is deleted by code that it calls, it is possible for `caller` to see a stack frame belonging to that deleted package. `caller` could crash if the stash’s memory address was reused for a scalar and a substitution was performed on the same scalar [perl #113486].
- `UNIVERSAL::can` no longer treats its first argument differently depending on whether it is a string or number internally.
- `open` with `<&` for the mode checks to see whether the third argument is a number, in determining whether to treat it as a file descriptor or a handle name. Magical variables like `$1` were always failing the numeric check and being treated as handle names.

- `warn`'s handling of magical variables (`$1`, ties) has undergone several fixes. `FETCH` is only called once now on a tied argument or a tied `$@` [perl #97480]. Tied variables returning objects that stringify as "" are no longer ignored. A tied `$@` that happened to return a reference the *previous* time it was used is no longer ignored.
- `warn ""`no w treats `$@` with a number in it the same way, regardless of whether it happened via `$@=3` or `$@="3"`. It used to ignore the former. Now it appends ‘‘\t...caught’’, as it has always done with `$@="3"`.
- Numeric operators on magical variables (e.g., `$1+1`) used to use floating point operations even where integer operations were more appropriate, resulting in loss of accuracy on 64-bit platforms [perl #109542].
- Unary negation no longer treats a string as a number if the string happened to be used as a number at some point. So, if `$x` contains the string ‘‘dogs’’, `-$x` returns ‘‘-dogs’’ even if `$y=0+$x` has happened at some point.
- In Perl v5.14, `-'-10'` was fixed to return ‘‘10’’, not ‘‘+10’’. But magical variables (`$1`, ties) were not fixed till now [perl #57706].
- Unary negation now treats strings consistently, regardless of the internal `UTF8` flag.
- A regression introduced in Perl v5.16.0 involving `tr/`*`SEARCHLIST`*`/REPLACEMENTLIST/` has been fixed. Only the first instance is supposed to be meaningful if a character appears more than once in *`SEARCHLIST`*. Under some circumstances, the final instance was overriding all earlier ones. [perl #113584]
- Regular expressions like `qr/\87/` previously silently inserted a NUL character, thus matching as if it had been written `qr/\00087/`. Now it matches as if it had been written as `qr/87/`, with a message that the sequence `"\8"` is unrecognized.
- `__SUB__` now works in special blocks (`BEGIN`, `END`, etc.).
- Thread creation on Windows could theoretically result in a crash if done inside a `BEGIN` block. It still does not work properly, but it no longer crashes [perl #111610].
- `\&{''}` (with the empty string) now autovivifies a stub like any other sub name, and no longer produces the ‘‘Unable to create sub’’ error [perl #94476].
- A regression introduced in v5.14.0 has been fixed, in which some calls to the `re` module would clobber `$_` [perl #113750].
- `do FILE`no w always either sets or clears `$@`, even when the file can't be read. This ensures that testing `$@` first (as recommended by the documentation) always returns the correct result.
- The array iterator used for the `each @array` construct is now correctly reset when `@array` is cleared [perl #75596]. This happens, for example, when the array is globally assigned to, as in `@array = (...)`, but not when its **values** are assigned to. In terms of the XS API, it means that `av_clear()` will now reset the iterator.

  This mirrors the behaviour of the hash iterator when the hash is cleared.
- `$class->can`, `$class->isa`, and `$class->DOES` now return correct results, regardless of whether that package referred to by `$class` exists [perl #47113].
- Arriving signals no longer clear `$@` [perl #45173].
- Allow `my ()` declarations with an empty variable list [perl #113554].
- During parsing, subs declared after errors no longer leave stubs [perl #113712].
- Closures containing no string evals no longer hang on to their containing subroutines, allowing variables closed over by outer subroutines to be freed when the outer sub is freed, even if the inner sub still exists [perl #89544].

- Duplication of in-memory filehandles by opening with a ‘‘<&=’’ or ‘‘>&=’’ mode stopped working properly in v5.16.0. It was causing the new handle to reference a different scalar variable. This has been fixed [perl #113764].
- `qr//` expressions no longer crash with custom regular expression engines that do not set `offs` at regular expression compilation time [perl #112962].
- `delete local`no longer crashes with certain magical arrays and hashes [perl #112966].
- `local` on elements of certain magical arrays and hashes used not to arrange to have the element deleted on scope exit, even if the element did not exist before `local`.
- `scalar(write)` no longer returns multiple items [perl #73690].
- String to floating point conversions no longer misparse certain strings under `use locale` [perl #109318].
- `@INC` filters that die no longer leak memory [perl #92252].
- The implementations of overloaded operations are now called in the correct context. This allows, among other things, being able to properly override `<>` [perl #47119].
- Specifying only the `fallback` key when calling `use overload` now behaves properly [perl #113010].
- `sub foo { my $a = 0; while ($a) { ... } }`and `sub foo { while (0) { ... } }`no w return the same thing [perl #73618].
- String negation now behaves the same under `use integer;` as it does without [perl #113012].
- `chr` now returns the Unicode replacement character (U+FFFD) for -1, regardless of the internal representation. -1 used to wrap if the argument was tied or a string internally.
- Using a `format` after its enclosing sub was freed could crash as of perl v5.12.0, if the format referenced lexical variables from the outer sub.
- Using a `format` after its enclosing sub was undefined could crash as of perl v5.10.0, if the format referenced lexical variables from the outer sub.
- Using a `format` defined inside a closure, which format references lexical variables from outside, never really worked unless the `write` call was directly inside the closure. In v5.10.0 it even started crashing. Now the copy of that closure nearest the top of the call stack is used to find those variables.
- Formats that close over variables in special blocks no longer crash if a stub exists with the same name as the special block before the special block is compiled.
- The parser no longer gets confused, treating `eval foo ()` as a syntax error if preceded by `print;` [perl #16249].
- The return value of `syscall` is no longer truncated on 64-bit platforms [perl #113980].
- Constant folding no longer causes `print 1 ? FOO : BAR` to print to the FOO handle [perl #78064].
- `do subname`no w calls the named subroutine and uses the file name it returns, instead of opening a file named ‘‘subname’’.
- Subroutines looked up by rv2cv check hooks (registered by XS modules) are now taken into consideration when determining whether `foo bar` should be the sub call `foo(bar)` or the method call `"bar"->foo`.
- `CORE::foo::bar` is no longer treated specially, allowing global overrides to be called directly via `CORE::GLOBAL::uc(...)` [perl #113016].
- Calling an undefined sub whose typeglob has been undefined now produces the customary ‘‘Undefined subroutine called’’ error, instead of ‘‘Not a CODE reference’’.

- Two bugs involving `@ISA` have been fixed. `*ISA = *glob_without_array` and `undef *ISA; @{*ISA}`w ould prevent future modifications to `@ISA` from updating the internal caches used to look up methods. The *glob_without_array case was a regression from Perl v5.12.
- Regular expression optimisations sometimes caused `$` with `/m` to produce failed or incorrect matches [perl #114068].
- `__SUB__` now works in a `sort` block when the enclosing subroutine is predeclared with `sub foo;` syntax [perl #113710].
- Unicode properties only apply to Unicode code points, which leads to some subtleties when regular expressions are matched against above-Unicode code points. There is a warning generated to draw your attention to this. However, this warning was being generated inappropriately in some cases, such as when a program was being parsed. Non-Unicode matches such as `\w` and `[:word:]` should not generate the warning, as their definitions don't limit them to apply to only Unicode code points. Now the message is only generated when matching against `\p{}` and `\P{}`. There remains a bug, [perl #114148], for the very few properties in Unicode that match just a single code point. The warning is not generated if they are matched against an above-Unicode code point.
- Uninitialized warnings mentioning hash elements would only mention the element name if it was not in the first bucket of the hash, due to an off-by-one error.
- A regular expression optimizer bug could cause multiline "^" to behave incorrectly in the presence of line breaks, such that `"/\n\n" =~ m#\A(?:^/$)#im` would not match [perl #115242].
- Failed `fork` in list context no longer corrupts the stack. `@a = (1, 2, fork, 3)` used to gobble up the 2 and assign `(1, undef, 3)` if the `fork` call failed.
- Numerous memory leaks have been fixed, mostly involving tied variables that die, regular expression character classes and code blocks, and syntax errors.
- Assigning a regular expression (`${qr//}`) to a variable that happens to hold a floating point number no longer causes assertion failures on debugging builds.
- Assigning a regular expression to a scalar containing a number no longer causes subsequent numification to produce random numbers.
- Assigning a regular expression to a magic variable no longer wipes away the magic. This was a regression from v5.10.
- Assigning a regular expression to a blessed scalar no longer results in crashes. This was also a regression from v5.10.
- Regular expression can now be assigned to tied hash and array elements with flattening into strings.
- Numifying a regular expression no longer results in an uninitialized warning.
- Negative array indices no longer cause EXISTS methods of tied variables to be ignored. This was a regression from v5.12.
- Negative array indices no longer result in crashes on arrays tied to non-objects.
- `$byte_overload .= $utf8`no longer results in doubly-encoded UTF-8 if the left-hand scalar happened to have produced a UTF-8 string the last time overloading was invoked.
- `goto &sub`no w uses the current value of `@_`, instead of using the array the subroutine was originally called with. This means `local @_ = (...); goto &sub` now works [perl #43077].
- If a debugger is invoked recursively, it no longer stomps on its own lexical variables. Formerly under recursion all calls would share the same set of lexical variables [perl #115742].
- `*_{ARRAY}` returned from a subroutine no longer spontaneously becomes empty.
- When using `say` to print to a tied filehandle, the value of `$\` is correctly localized, even if it was previously undef. [perl #119927]

Known Problems

- UTF8-flagged strings in %ENV on HP-UX 11.00 are buggy

  The interaction of UTF8-flagged strings and %ENV on HP-UX 11.00 is currently dodgy in some not-yet-fully-diagnosed way. Expect test failures in *t/op/magic.t*, followed by unknown behavior when storing wide characters in the environment.

Obituary

Hojung Yoon (AMORETTE), 24, of Seoul, South Korea, went to his long rest on May 8, 2013 with llama figurine and autographed TIMTOADY card. He was a brilliant young Perl 5 & 6 hacker and a devoted member of Seoul.pm. He programmed Perl, talked Perl, ate Perl, and loved Perl. We believe that he is still programming in Perl with his broken IBM laptop somewhere. He will be missed.

Acknowledgements

Perl v5.18.0 represents approximately 12 months of development since Perl v5.16.0 and contains approximately 400,000 lines of changes across 2,100 files from 113 authors.

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 v5.18.0:

Aaron Crane, Aaron Trevena, Abhijit Menon-Sen, Adrian M. Enache, Alan Haggai Alavi, Alexandr Ciornii, Andrew Tam, Andy Dougherty, Anton Nikishaev, Aristotle Pagaltzis, Augustina Blair, Bob Ernst, Brad Gilbert, Breno G. de Oliveira, Brian Carlson, Brian Fraser, Charlie Gonzalez, Chip Salzenberg, Chris 'BinGOs' Williams, Christian Hansen, Colin Kuskie, Craig A. Berry, Dagfinn Ilmari Mannsåker, Daniel Dragan, Daniel Perrett, Darin McBride, Dave Rolsky, David Golden, David Leadbeater, David Mitchell, David Nicol, Dominic Hargreaves, E. Choroba, Eric Brine, Evan Miller, Father Chrysostomos, Florian Ragwitz, François Perrad, George Greer, Goro Fuji, H.Merijn Brand, Herbert Breunung, Hugo van der Sanden, Igor Zaytsev, James E Keenan, Jan Dubois, Jasmine Ahuja, Jerry D. Hedden, Jess Robinson, Jesse Luehrs, Joaquin Ferrero, Joel Berger, John Goodyear, John Peacock, Karen Etheridge, Karl Williamson, Karthik Rajagopalan, Kent Fredric, Leon Timmermans, Lucas Holt, Lukas Mai, Marcus Holland-Moritz, Markus Jansen, Martin Hasch, Matthew Horsfall, Max Maischein, Michael G Schwern, Michael Schroeder, Moritz Lenz, Nicholas Clark, Niko Tyni, Oleg Nesterov, Patrik Hägglund, Paul Green, Paul Johnson, Paul Marquess, Peter Martini, Rafael Garcia-Suarez, Reini Urban, Renee Baecker, Rhesa Rozendaal, Ricardo Signes, Robin Barker, Ronald J. Kimball, Ruslan Zakirov, Salvador Fandiño, Sawyer X, Scott Lanning, Sergey Alekseev, Shawn M Moore, Shirakata Kentaro, Shlomi Fish, Sisyphus, Smylers, Steffen Müller, Steve Hay, Steve Peters, Steven Schubiger, Sullivan Beck, Sven Strickroth, Sébastien Aperghis-Tramoni, Thomas Sibley, Tobias Leich, Tom Wyant, Tony Cook, Vadim Konovalov, Vincent Pit, Volker Schatz, Walt Mankowski, Yves Orton, Zefram.

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 http://rt.perl.org/perlbug/ http://www.perl.org/ , the Perl Home Page.

If you believe you have an unreported bug, please run the perlbug(1) 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.