NAME

perl5160delta - what is new for perl v5.16.0

DESCRIPTION

This document describes differences between the 5.14.0 release and the 5.16.0 release.

If you are upgrading from an earlier release such as 5.12.0, first read perl5140delta, which describes differences between 5.12.0 and 5.14.0.

Some bug fixes in this release have been backported to later releases of 5.14.x. Those are indicated with the 5.14.x version in parentheses.

Notice

With the release of Perl 5.16.0, the 5.12.x series of releases is now out of its support period. There may be future 5.12.x releases, but only in the event of a critical security issue. Users of Perl 5.12 or earlier should consider upgrading to a more recent release of Perl.

This policy is described in greater detail in perlpolicy.

Core Enhancements

`use` *`VERSION`*

As of this release, version declarations like `use v5.16` now disable all features before enabling the new feature bundle. This means that the following holds true:

```
use 5.016;
# only 5.16 features enabled here
use 5.014;
# only 5.14 features enabled here (not 5.16)
```

`use v5.12`and higher continue to enable strict, b ut explicit `use strict` and `no strict` now override the version declaration, even when they come first:

```
no strict;
use 5.012;
# no strict here
```

There is a new ‘‘:default’’ feature bundle that represents the set of features enabled before any version declaration or `use feature` has been seen. Version declarations below 5.10 now enable the ‘‘:default’’ feature set. This does not actually change the behavior of `use v5.8`, because features added to the ‘‘:default’’ set are those that were traditionally enabled by default, before they could be turned off.

`no feature`no w resets to the default feature set. To disable all features (which is likely to be a pretty special-purpose request, since it presumably won’t match any named set of semantics) you can now write `no feature ':all'`.

`$[` is now disabled under `use v5.16`. It is part of the default feature set and can be turned on or off explicitly with `use feature 'array_base'`.

`__SUB__`

The new `__SUB__` token, available under the `current_sub` feature (see feature) or `use v5.16`, returns a reference to the current subroutine, making it easier to write recursive closures.

New and Improved Built-ins

More consistent `eval`

The `eval` operator sometimes treats a string argument as a sequence of characters and sometimes as a sequence of bytes, depending on the internal encoding. The internal encoding is not supposed to make any difference, but there is code that relies on this inconsistency.

The new `unicode_eval` and `evalbytes` features (enabled under `use 5.16.0`) resolve this. The `unicode_eval` feature causes `eval $string` to treat the string always as Unicode. The `evalbytes` features provides a function, itself called `evalbytes`, which evaluates its argument always as a string of bytes.

These features also fix oddities with source filters leaking to outer dynamic scopes.

See feature for more detail.

`substr` lvalue revamp

When `substr` is called in lvalue or potential lvalue context with two or three arguments, a special lvalue scalar is returned that modifies the original string (the first argument) when assigned to.

Previously, the offsets (the second and third arguments) passed to `substr` would be converted immediately to match the string, negative offsets being translated to positive and offsets beyond the end of the string being truncated.

Now, the offsets are recorded without modification in the special lvalue scalar that is returned, and the original string is not even looked at by `substr` itself, but only when the returned lvalue is read or modified.

These changes result in an incompatible change:

If the original string changes length after the call to `substr` but before assignment to its return value, negative offsets will remember their position from the end of the string, affecting code like this:

```
my $string = "string";
my $lvalue = \substr $string, -4, 2;
print $$lvalue, "\n"; # prints "ri"
$string = "bailing twine";
print $$lvalue, "\n"; # prints "wi"; used to print "il"
```

The same thing happens with an omitted third argument. The returned lvalue will always extend to the end of the string, even if the string becomes longer.

Since this change also allowed many bugs to be fixed (see "The `substr` operator"), and since the behavior of negative offsets has never been specified, the change was deemed acceptable.

Return value of `tied`

The value returned by `tied` on a tied variable is now the actual scalar that holds the object to which the variable is tied. This lets ties be weakened with `Scalar::Util::weaken(tied $tied_variable)`.

Unicode Support

Supports (almost) Unicode 6.1

Besides the addition of whole new scripts, and new characters in existing scripts, this new version of Unicode, as always, makes some changes to existing characters. One change that may trip up some applications is that the General Category of two characters in the Latin-1 range, PILCROW SIGN and SECTION SIGN, has been changed from Other_Symbol to Other_Punctuation. The same change has been made for a character in each of Tibetan, Ethiopic, and Aegean. The code points U+3248..U+324F (CIRCLED NUMBER TEN ON BLACK SQUARE through CIRCLED NUMBER EIGHTY ON BLACK SQUARE) have had their General Category changed from Other_Symbol to Other_Numeric. The Line Break property has changes for Hebrew and Japanese; and because of other changes in 6.1, the Perl regular expression construct `\X` now works differently for some characters in Thai and Lao.

New aliases (synonyms) have been defined for many property values; these, along with the previously existing ones, are all cross-indexed in perluniprops.

The return value of `charnames::viacode()` is affected by other changes:

```
Code point Old Name New Name
U+000A LINE FEED (LF) LINE FEED
U+000C FORM FEED (FF) FORM FEED
U+000D CARRIAGE RETURN (CR) CARRIAGE RETURN
U+0085 NEXT LINE (NEL) NEXT LINE
U+008E SINGLE-SHIFT 2 SINGLE-SHIFT-2
U+008F SINGLE-SHIFT 3 SINGLE-SHIFT-3
U+0091 PRIVATE USE 1 PRIVATE USE-1
U+0092 PRIVATE USE 2 PRIVATE USE-2
U+2118 SCRIPT CAPITAL P WEIERSTRASS ELLIPTIC FUNCTION
```

Perl will accept any of these names as input, but `charnames::viacode()` now returns the new name of each pair. The change for U+2118 is considered by Unicode to be a correction, that is the original name was a mistake (but again, it will remain forever valid to use it to refer to U+2118). But most of these changes are the fallout of the mistake Unicode 6.0 made in naming a character used in Japanese cell phones to be ‘‘BELL’’, which conflicts with the longstanding industry use of (and Unicode’s recommendation to use) that name to mean the ASCII control character at U+0007. Therefore, that name has been deprecated in Perl since v5.14, and any use of it will raise a warning message (unless turned off). The name ‘‘ALERT’’ is now the preferred name for this code point, with ‘‘BEL’’ an acceptable short form. The name for the new cell phone character, at code point U+1F514, remains undefined in this version of Perl (hence we don’t implement quite all of Unicode 6.1), but starting in v5.18, BELL will mean this character, and not U+0007.

Unicode has taken steps to make sure that this sort of mistake does not happen again. The Standard now includes all generally accepted names and abbreviations for control characters, whereas previously it didn’t (though there were recommended names for most of them, which Perl used). This means that most of those recommended names are now officially in the Standard. Unicode did not recommend names for the four code points listed above between U+008E and U+008F, and in standardizing them Unicode subtly changed the names that Perl had previously given them, by replacing the final blank in each name by a hyphen. Unicode also officially accepts names that Perl had deprecated, such as FILE SEPARATOR. Now the only deprecated name is BELL. Finally, Perl now uses the new official names instead of the old (now considered obsolete) names for the first four code points in the list above (the ones which have the parentheses in them).

Now that the names have been placed in the Unicode standard, these kinds of changes should not happen again, though corrections, such as to U+2118, are still possible.

Unicode also added some name abbreviations, which Perl now accepts: SP for SPACE; TAB for CHARACTER TABULATION; NEW LINE, END OF LINE, NL, and EOL for LINE FEED; LOCKING-SHIFT ONE for SHIFT OUT; LOCKING-SHIFT ZERO for SHIFT IN; and ZWNBSP for ZERO WIDTH NO-BREAK SPACE.

More details on this version of Unicode are provided in <http://www.unicode.org/versions/Unicode6.1.0/>.

`use charnames`is no long er needed for `\N{name}`

When `\N{name}` is encountered, the `charnames` module is now automatically loaded when needed as if the `:full` and `:short` options had been specified. See charnames for more information.

`\N{...}` can now have Unicode loose name matching

This is described in the `charnames` item in ‘‘Updated Modules and Pragmata’’ below.

Unicode Symbol Names

Perl now has proper support for Unicode in symbol names. It used to be that `*{$foo}` would ignore the internal UTF8 flag and use the bytes of the underlying representation to look up the symbol. That meant that `*{"\x{100}"}` and `*{"\xc4\x80"}` would return the same thing. All these parts of Perl have been fixed to account for Unicode:

- Method names (including those passed to `use overload`)

- Typeglob names (including names of variables, subroutines, and filehandles)
- Package names
- `goto`
- Symbolic dereferencing
- Second argument to `bless()` and `tie()`
- Return value of `ref()`
- Subroutine prototypes
- Attributes
- Various warnings and error messages that mention variable names or values, methods, etc.

In addition, a parsing bug has been fixed that prevented `*{é}` from implicitly quoting the name, but instead interpreted it as `*{+é}`, which would cause a strict violation.

`*{"*a::b"}` automatically strips off the * if it is followed by an ASCII letter. That has been extended to all Unicode identifier characters.

One-character non-ASCII non-punctuation variables (like `$é`) are now subject to ‘‘Used only once’’ warnings. They used to be exempt, as they were treated as punctuation variables.

Also, single-character Unicode punctuation variables (like `$‰`) are now supported [perl #69032].

Improved ability to mix locales and Unicode, including UTF-8 locales

An optional parameter has been added to `use locale`

```
 use locale ':not_characters';
```

which tells Perl to use all but the `LC_CTYPE` and `LC_COLLATE` portions of the current locale. Instead, the character set is assumed to be Unicode. This lets locales and Unicode be seamlessly mixed, including the increasingly frequent UTF-8 locales. When using this hybrid form of locales, the `:locale` layer to the open pragma can be used to interface with the file system, and there are CPAN modules available for ARGV and environment variable conversions.

Full details are in perllocale.

New function `fc` and corresponding escape sequence `\F` for Unicode foldcase

Unicode foldcase is an extension to lowercase that gives better results when comparing two strings case-insensitively. It has long been used internally in regular expression `/i` matching. Now it is available explicitly through the new `fc` function call (enabled by `"usefeature'fc'"`, or `use v5.16`, or explicitly callable via `CORE::fc` or through the new `\F` sequence in double-quotish strings.

Full details are in ‘‘fc’’ in perlfunc.

The Unicode `Script_Extensions` property is now supported.

New in Unicode 6.0, this is an improved `Script` property. Details are in ‘‘Scripts’’ in perlunicode.

XS Changes

Improved typemaps for Some Builtin Types

Most XS authors will know there is a longstanding bug in the OUTPUT typemap for T_AVREF (`AV*`), T_HVREF (`HV*`), T_CVREF (`CV*`), and T_SVREF (`SVREF` or `\$foo`) that requires manually decrementing the reference count of the return value instead of the typemap taking care of this. For backwards-compatibility, this cannot be changed in the default typemaps. But we now provide additional typemaps `T_AVREF_REFCOUNT_FIXED`, etc. that do not exhibit this bug. Using them in your extension is as simple as having one line in your `TYPEMAP` section:

```
 HV* T_HVREF_REFCOUNT_FIXED
```

`is_utf8_char()`

The XS-callable function `is_utf8_char()`, when presented with malformed UTF-8 input, can read up to 12 bytes beyond the end of the string. This cannot be fixed without changing its API, and so its use is now deprecated. Use `is_utf8_char_buf()` (described just below) instead.

Added `is_utf8_char_buf()`

This function is designed to replace the deprecated "*is_utf8_char()*" function. It includes an extra parameter to make sure it doesn't read past the end of the input buffer.

Other `is_utf8_foo()` functions, as well as `utf8_to_foo()`, etc.

Most other XS-callable functions that take UTF-8 encoded input implicitly assume that the UTF-8 is valid (not malformed) with respect to buffer length. Do not do things such as change a character's case or see if it is alphanumeric without first being sure that it is valid UTF-8. This can be safely done for a whole string by using one of the functions`is_utf8_string()`, `is_utf8_string_loc()`, and `is_utf8_string_loclen()`.

New Pad API

Many new functions have been added to the API for manipulating lexical pads. See "Pad Data Structures" in perlapi(1) for more information.

Changes to Special Variables

$$ can be assigned to

`$$` was made read-only in Perl 5.8.0. But only sometimes: `local $$` would make it writable again. Some CPAN modules were using `local $$` or XS code to bypass the read-only check, so there is no reason to keep `$$` read-only. (This change also allowed a bug to be fixed while maintaining backward compatibility.)

`$^X` converted to an absolute path on FreeBSD, OS X and Solaris

`$^X` is now converted to an absolute path on OS X, FreeBSD (without needing */proc* mounted) and Solaris 10 and 11. This augments the previous approach of using */proc* on Linux, FreeBSD, and NetBSD (in all cases, where mounted).

This makes relocatable perl installations more useful on these platforms. (See "Relocatable `@INC`" in *INSTALL*)

Debugger Changes

Features inside the debugger

The current Perl's feature bundle is now enabled for commands entered in the interactive debugger.

*New option for the debugger's **t** command*

The **t** command in the debugger, which toggles tracing mode, now accepts a numeric argument that determines how many levels of subroutine calls to trace.

`enable` and `disable`

The debugger now has `disable` and `enable` commands for disabling existing breakpoints and re-enabling them. See perldebug.

Breakpoints with file names

The debugger's "b" command for setting breakpoints now lets a line number be prefixed with a file name. See "b [file]:[line] [condition]" in perldebug.

The `CORE` Namespace

The `CORE::` prefix

The `CORE::` prefix can now be used on keywords enabled by feature.pm, even outside the scope of `use feature`.

Subroutines in the `CORE` namespace

Many Perl keywords are now available as subroutines in the CORE namespace. This lets them be aliased:

```
BEGIN { *entangle = \&CORE::tie }
entangle $variable, $package, @args;
```

And for prototypes to be bypassed:

```
sub mytie(\[%$*@]$@) {
my ($ref, $pack, @args) = @_;
... do something ...
goto &CORE::tie;
}
```

Some of these cannot be called through references or via `&foo` syntax, but must be called as barewords.

See CORE for details.

Other Changes

Anonymous handles

Automatically generated file handles are now named __ANONIO__ when the variable name cannot be determined, rather than $__ANONIO__.

Autoloaded sort Subroutines

Custom sort subroutines can now be autoloaded [perl #30661]:

```
sub AUTOLOAD { ... }
@sorted = sort foo @list; # uses AUTOLOAD
```

`continue` no longer requires the "switch" feature

The `continue` keyword has two meanings. It can introduce a `continue` block after a loop, or it can exit the current `when` block. Up to now, the latter meaning was valid only with the "switch" feature enabled, and was a syntax error otherwise. Since the main purpose of feature.pm is to avoid conflicts with user-defined subroutines, there is no reason for `continue` to depend on it.

DTrace probes for interpreter phase change

The `phase-change` probes will fire when the interpreter's phase changes, which tracks the `${^GLOBAL_PHASE}` variable. `arg0` is the new phase name; `arg1` is the old one. This is useful for limiting your instrumentation to one or more of: compile time, run time, or destruct time.

`__FILE__()` Syntax

The `__FILE__`, `__LINE__` and `__PACKAGE__` tokens can now be written with an empty pair of parentheses after them. This makes them parse the same way as `time`, `fork` and other built-in functions.

The `\$` prototype accepts any scalar lvalue

The `\$` and `\[$]` subroutine prototypes now accept any scalar lvalue argument. Previously they accepted only scalars beginning with `$` and hash and array elements. This change makes them consistent with the way the built-in `read` and `recv` functions (among others) parse their arguments. This means that one can override the built-in functions with custom subroutines that parse their arguments the same way.

_ in subroutine prototypes

The _ character in subroutine prototypes is now allowed before `@` or `%`.

Security

Use `is_utf8_char_buf()` and not `is_utf8_char()`

The latter function is now deprecated because its API is insufficient to guarantee that it doesn't read (up to 12 bytes in the worst case) beyond the end of its input string. See *is_utf8_char_buf()*.

Malformed UTF−8 input could cause attempts to read beyond the end of the buffer

Two new XS-accessible functions, `utf8_to_uvchr_buf()` and `utf8_to_uvuni_buf()` are now available to prevent this, and the Perl core has been converted to use them. See "Internal Changes".

`File::Glob::bsd_glob()` memory error with GLOB_ALTDIRFUNC (CVE-2011-2728).

Calling `File::Glob::bsd_glob` with the unsupported flag GLOB_ALTDIRFUNC would cause an access violation / segfault. A Perl program that accepts a flags value from an external source could expose itself to denial of service or arbitrary code execution attacks. There are no known exploits in the wild. The problem has been corrected by explicitly disabling all unsupported flags and setting unused function pointers to null. Bug reported by Clément Lecigne. (5.14.2)

Privileges are now set correctly when assigning to `$(`

A hypothetical bug (probably unexploitable in practice) because the incorrect setting of the effective group ID while setting `$(` has been fixed. The bug would have affected only systems that have `setresgid()` but not `setregid()`, but no such systems are known to exist.

Deprecations

Don't read the Unicode data base files in *lib/unicore*

It is now deprecated to directly read the Unicode data base files. These are stored in the *lib/unicore* directory. Instead, you should use the new functions in Unicode::UCD. These provide a stable API, and give complete information.

Perl may at some point in the future change or remove these files. The file which applications were most likely to have used is *lib/unicore/ToDigit.pl*. "*prop_invmap()*" in Unicode::UCD can be used to get at its data instead.

XS functions `is_utf8_char()`, `utf8_to_uvchr()` and `utf8_to_uvuni()`

This function is deprecated because it could read beyond the end of the input string. Use the new *is_utf8_char_buf()*, `utf8_to_uvchr_buf()` and `utf8_to_uvuni_buf()` instead.

Future Deprecations

This section serves as a notice of features that are *likely* to be removed or deprecated in the next release of perl (5.18.0). If your code depends on these features, you should contact the Perl 5 Porters via the mailing list <http://lists.perl.org/list/perl5-porters.html> or perlbug to explain your use case and inform the deprecation process.

Core Modules

These modules may be marked as deprecated *from the core*. This only means that they will no longer be installed by default with the core distribution, but will remain available on the CPAN.

- CPANPLUS
- Filter::Simple
- PerlIO::mmap
- Pod::LaTeX
- Pod::Parser
- SelfLoader
- Text::Soundex
- Thread.pm

Platforms with no supporting programmers

These platforms will probably have their special build support removed during the 5.17.0 development series.

- BeOS
- djgpp
- dgux
- EPOC
- MPE/iX

- Rhapsody
- UTS
- VM/ESA

Other Future Deprecations

- Swapping of $< and $>

  For more information about this future deprecation, see the relevant RT ticket <https://rt.perl.org/rt3/Ticket/Display.html?id=96212>.

- sfio, stdio

  Perl supports being built without PerlIO proper, using a stdio or sfio wrapper instead. A perl build like this will not support IO layers and thus Unicode IO, making it rather handicapped.

  PerlIO supports a `stdio` layer if stdio use is desired, and similarly a sfio layer could be produced.

- Unescaped literal `"{"` in regular expressions.

  Starting with v5.20, it is planned to require a literal `"{"` to be escaped, for example by preceding it with a backslash. In v5.18, a deprecated warning message will be emitted for all such uses. This affects only patterns that are to match a literal `"{"`. Other uses of this character, such as part of a quantifier or sequence as in those below, are completely unaffected:

  ```
  /foo{3,5}/
  /\p{Alphabetic}/
  /\N{DIGIT ZERO}
  ```

  Removing this will permit extensions to Perl's pattern syntax and better error checking for existing syntax. See "Quantifiers" in perlre(1) for an example.

- 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.

Incompatible Changes

Special blocks called in void context

Special blocks (`BEGIN`, `CHECK`, `INIT`, `UNITCHECK`, `END`) are now called in void context. This avoids wasteful copying of the result of the last statement [perl #108794].

The `overloading` pragma and regexp objects

With `no overloading`, regular expression objects returned by `qr//` are now stringified as "Regexp=REGEXP(0xbe600d)" instead of the regular expression itself [perl #108780].

Two XS typemap Entries removed

Two presumably unused XS typemap entries have been removed from the core typemap: T_DATAUNIT and T_CALLBACK. If you are, against all odds, a user of these, please see the instructions on how to restore them in perlxstypemap.

Unicode 6.1 has incompatibilities with Unicode 6.0

These are detailed in "Supports (almost) Unicode 6.1" above. You can compile this version of Perl to use Unicode 6.0. See "Hacking Perl to work on earlier Unicode versions (for very serious hackers only)" in perlunicode.

Borland compiler

All support for the Borland compiler has been dropped. The code had not worked for a long time anyway.

Certain deprecated Unicode properties are no longer supported by default

Perl should never have exposed certain Unicode properties that are used by Unicode internally and not meant to be publicly available. Use of these has generated deprecated warning messages since Perl 5.12. The removed properties are Other_Alphabetic, Other_Default_Ignorable_Code_Point, Other_Grapheme_Extend, Other_ID_Continue, Other_ID_Start, Other_Lowercase, Other_Math, and Other_Uppercase.

Perl may be recompiled to include any or all of them; instructions are given in "Unicode character properties that are NOT accepted by Perl" in perluniprops.

Dereferencing IO thingies as typeglobs

The `*{...}` operator, when passed a reference to an IO thingy (as in `*{*STDIN{IO}}`), creates a new typeglob containing just that IO object. Previously, it would stringify as an empty string, but some operators would treat it as undefined, producing an "uninitialized" warning. Now it stringifies as __ANONIO__ [perl #96326].

User-defined case-changing operations

This feature was deprecated in Perl 5.14, and has now been removed. The CPAN module Unicode::Casing provides better functionality without the drawbacks that this feature had, as are detailed in the 5.14 documentation: <http://perldoc.perl.org/5.14.0/perlunicode.html#User-Defined-Case-Mappings-%28for-serious-hackers-only%29>

XSUBs are now 'static'

XSUB C functions are now 'static', that is, they are not visible from outside the compilation unit. Users can use the new `XS_EXTERNAL(name)` and `XS_INTERNAL(name)` macros to pick the desired linking behavior. The ordinary`XS(name)` declaration for XSUBs will continue to declare non-' static' XSUBs for compatibility, but the XS compiler, ExtUtils::ParseXS (`xsubpp`) will emit 'static' XSUBs by default. ExtUtils::ParseXS's behavior can be reconfigured from XS using the `EXPORT_XSUB_SYMBOLS` keyword. See perlxs(1) for details.

Weakening read-only references

Weakening read-only references is no longer permitted. It should never have worked anyway, and could sometimes result in crashes.

Tying scalars that hold typeglobs

Attempting to tie a scalar after a typeglob was assigned to it would instead tie the handle in the typeglob's IO slot. This meant that it was impossible to tie the scalar itself. Similar problems affected `tied` and `untie`: `tied $scalar` would return false on a tied scalar if the last thing returned was a typeglob, and `untie $scalar`on such a tied scalar w ould do nothing.

We fixed this problem before Perl 5.14.0, but it caused problems with some CPAN modules, so we put in a deprecation cycle instead.

Now the deprecation has been removed and this bug has been fixed. So `tie $scalar` will always tie the scalar, not the handle it holds. To tie the handle, use `tie *$scalar` (with an explicit asterisk). The same applies to `tied *$scalar` and `untie *$scalar`.

IPC::Open3 no longer provides `xfork()`, `xclose_on_exec()` and `xpipe_anon()`

All three functions were private, undocumented, and unexported. They do not appear to be used by any code on CPAN. Two have been inlined and one deleted entirely.

$$ no longer caches PID

Previously, if one called *fork(3)* from C, Perl's notion of $$ could go out of sync with what *getpid()* returns. By always fetching the value of $$ via *getpid()*, this potential bug is eliminated. Code that depends on the caching behavior will break. As described in Core Enhancements, $$ is now writable, but it will be reset during a fork.

$$ and `getppid()` no longer emulate POSIX semantics under LinuxThreads

The POSIX emulation of $$ and `getppid()` under the obsolete LinuxThreads implementation has been removed. This only impacts users of Linux 2.4 and users of Debian GNU/kFreeBSD up to and including 6.0, not the vast majority of Linux installations that use NPTL threads.

This means that `getppid()`, like $$, is now always guaranteed to return the OS's idea of the current state of the process, not perl's cached version of it.

See the documentation for $$ for details.

`$<`, `$>`, `$(` and `$)` are no longer cached

Similarly to the changes to `$$` and `getppid()`, the internal caching of `$<`, `$>`, `$(` and `$)` has been removed.

When we cached these values our idea of what they were would drift out of sync with reality if someone (e.g., someone embedding perl) called `sete?[ug]id()` without updating `PL_e?[ug]id`. Having to deal with this complexity wasn't worth it given how cheap the `gete?[ug]id()` system call is.

This change will break a handful of CPAN modules that use the XS-level `PL_uid`, `PL_gid`, `PL_euid` or `PL_egid` variables.

The fix for those breakages is to use `PerlProc_gete?[ug]id()` to retrieve them (e.g., `PerlProc_getuid()`), and not to assign to `PL_e?[ug]id` if you change the UID/GID/EUID/EGID. There is no longer any need to do so since perl will always retrieve the up-to-date version of those values from the OS.

Which Non-ASCII characters get quoted by `quotemeta` and `\Q` has changed

This is unlikely to result in a real problem, as Perl does not attach special meaning to any non-ASCII character, so it is currently irrelevant which are quoted or not. This change fixes bug [perl #77654] and brings Perl's behavior more into line with Unicode's recommendations. See ''quotemeta'' in perlfunc.

Performance Enhancements

- Improved performance for Unicode properties in regular expressions

  Matching a code point against a Unicode property is now done via a binary search instead of linear. This means for example that the worst case for a 1000 item property is 10 probes instead of 1000. This inefficiency has been compensated for in the past by permanently storing in a hash the results of a given probe plus the results for the adjacent 64 code points, under the theory that near-by code points are likely to be searched for. A separate hash was used for each mention of a Unicode property in each regular expression. Thus, `qr/\p{foo}abc\p{foo}/` would generate two hashes. Any probes in one instance would be unknown to the other, and the hashes could expand separately to be quite large if the regular expression were used on many different widely-separated code points. Now, however, there is just one hash shared by all instances of a given property. This means that if `\p{foo}` is matched against ''A'' in one regular expression in a thread, the result will be known immediately to all regular expressions, and the relentless march of using up memory is slowed considerably.

- Version declarations with the `use` keyword (e.g., `use 5.012`) are now faster, as they enable features without loading *feature.pm*.

- `local $_`is f aster now, as it no longer iterates through magic that it is not going to copy anyway.

- Perl 5.12.0 sped up the destruction of objects whose classes define empty `DESTROY` methods (to prevent autoloading), by simply not calling such empty methods. This release takes this optimization a step further, by not calling any `DESTROY` method that begins with a `return` statement. This can be useful for destructors that are only used for debugging:

  ```
  use constant DEBUG => 1;
  sub DESTROY { return unless DEBUG; ... }
  ```

  Constant-folding will reduce the first statement to `return;` if DEBUG is set to 0, triggering this optimization.

- Assigning to a variable that holds a typeglob or copy-on-write scalar is now much faster. Previously the typeglob would be stringified or the copy-on-write scalar would be copied before being clobbered.

- Assignment to `substr` in void context is now more than twice its previous speed. Instead of creating and returning a special lvalue scalar that is then assigned to, `substr` modifies the original string itself.

- `substr` no longer calculates a value to return when called in void context.

- Due to changes in File::Glob, Perl's `glob` function and its `<...>` equivalent are now much faster. The splitting of the pattern into words has been rewritten in C, resulting in speed-ups of 20% for some cases.

This does not affect `glob` on VMS, as it does not use File::Glob.

- The short-circuiting operators &&, `||`, and `//`, when chained (such as `$a || $b || $c`), are now considerably faster to short-circuit, due to reduced optree traversal.
- The implementation of `s///r` makes one fewer copy of the scalar's value.
- Recursive calls to lvalue subroutines in lvalue scalar context use less memory.

Modules and Pragmata

Deprecated Modules

Version::Requirements

Version::Requirements is now DEPRECATED, use CPAN::Meta::Requirements, which is a drop-in replacement. It will be deleted from perl.git blead in v5.17.0.

New Modules and Pragmata

- arybase — this new module implements the `$[` variable.
- PerlIO::mmap 0.010 has been added to the Perl core.

  The `mmap` PerlIO layer is no longer implemented by perl itself, but has been moved out into the new PerlIO::mmap module.

Updated Modules and Pragmata

This is only an overview of selected module updates. For a complete list of updates, run:

```
 $ corelist --diff 5.14.0 5.16.0
```

You can substitute your favorite version in place of 5.14.0, too.

- Archive::Extract has been upgraded from version 0.48 to 0.58.

  Includes a fix for FreeBSD to only use `unzip` if it is located in `/usr/local/bin`, as FreeBSD 9.0 will ship with a limited `unzip` in `/usr/bin`.
- Archive::Tar has been upgraded from version 1.76 to 1.82.

  Adjustments to handle files >8gb (>0777777777777 octal) and a feature to return the MD5SUM of files in the archive.
- base has been upgraded from version 2.16 to 2.18.

  `base` no longer sets a module's `$VERSION` to "-1" when a module it loads does not define a `$VERSION`. This change has been made because "-1" is not a valid version number under the new "lax" criteria used internally by `UNIVERSAL::VERSION` (See version for more on "lax" version criteria.)

  `base` no longer internally skips loading modules it has already loaded and instead relies on `require` to inspect `%INC`. This fixes a bug when `base` is used with code that clear `%INC` to force a module to be reloaded.
- Carp has been upgraded from version 1.20 to 1.26.

  It now includes last read filehandle info and puts a dot after the file and line number, just like errors from `die` [perl #106538].
- charnames has been updated from version 1.18 to 1.30.

  `charnames` can now be invoked with a new option, `:loose`, which is like the existing `:full` option, but enables Unicode loose name matching. Details are in "LOOSE MATCHES" in charnames.
- B::Deparse has been upgraded from version 1.03 to 1.14. This fixes numerous deparsing bugs.
- CGI has been upgraded from version 3.52 to 3.59.

  It uses the public and documented FCGI.pm API in CGI::Fast. CGI::Fast was using an FCGI API that was deprecated and removed from documentation more than ten years ago. Usage of this deprecated

API with FCGI >= 0.70 or FCGI <= 0.73 introduces a security issue. <https://rt.cpan.org/Public/Bug/Display.html?id=68380> <http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-2766>

Things that may break your code:

`url()` was fixed to return `PATH_INFO` when it is explicitly requested with either the `path=>1` or `path_info=>1` flag.

If your code is running under mod_rewrite (or compatible) and you are calling `self_url()` or you are calling `url()` and passing `path_info=>1`, these methods will actually be returning `PATH_INFO` now, as you have explicitly requested or `self_url()` has requested on your behalf.

The `PATH_INFO` has been omitted in such URLs since the issue was introduced in the 3.12 release in December, 2005.

This bug is so old your application may have come to depend on it or workaround it. Check for application before upgrading to this release.

Examples of affected method calls:

```
$q->url(-absolute => 1, -query => 1, -path_info => 1);
$q->url(-path=>1);
$q->url(-full=>1,-path=>1);
$q->url(-rewrite=>1,-path=>1);
$q->self_url();
```

We no longer read from STDIN when the Content-Length is not set, preventing requests with no Content-Length from sometimes freezing. This is consistent with the CGI RFC 3875, and is also consistent with CGI::Simple. However, the old behavior may have been expected by some command-line uses of CGI.pm.

In addition, the DELETE HTTP verb is now supported.

- Compress::Zlib has been upgraded from version 2.035 to 2.048.

  IO::Compress::Zip and IO::Uncompress::Unzip now have support for LZMA (method 14). There is a fix for a CRC issue in IO::Compress::Unzip and it supports Streamed Stored context now. And fixed a Zip64 issue in IO::Compress::Zip when the content size was exactly 0xFFFFFFFF.

- Digest::SHA has been upgraded from version 5.61 to 5.71.

  Added BITS mode to the addfile method and shasum. This makes partial-byte inputs possible via files/STDIN and lets shasum check all 8074 NIST Msg vectors, where previously special programming was required to do this.

- Encode has been upgraded from version 2.42 to 2.44.

  Missing aliases added, a deep recursion error fixed and various documentation updates.

  Addressed 'decode_xs n-byte heap-overflow' security bug in Unicode.xs (CVE-2011-2939). (5.14.2)

- ExtUtils::CBuilder updated from version 0.280203 to 0.280206.

  The new version appends CFLAGS and LDFLAGS to their Config.pm counterparts.

- ExtUtils::ParseXS has been upgraded from version 2.2210 to 3.16.

  Much of ExtUtils::ParseXS, the module behind the XS compiler `xsubpp`, was rewritten and cleaned up. It has been made somewhat more extensible and now finally uses strictures.

  The typemap logic has been moved into a separate module, ExtUtils::Typemaps. See "New Modules and Pragmata", above.

  For a complete set of changes, please see the ExtUtils::ParseXS changelog, available on the CPAN.

- File::Glob has been upgraded from version 1.12 to 1.17.

  On Windows, tilde (˜) expansion now checks the `USERPROFILE` environment variable, after checking `HOME`.

  It has a new `:bsd_glob` export tag, intended to replace `:glob`. Like `:glob` it overrides `glob` with a function that does not split the glob pattern into words, but, unlike `:glob`, it iterates properly in scalar context, instead of returning the last file.

  There are other changes affecting Perl's own `glob` operator (which uses File::Glob internally, except on VMS). See "Performance Enhancements" and "Selected Bug Fixes".

- FindBin updated from version 1.50 to 1.51.

  It no longer returns a wrong result if a script of the same name as the current one exists in the path and is executable.

- HTTP::Tiny has been upgraded from version 0.012 to 0.017.

  Added support for using `$ENV{http_proxy}` to set the default proxy host.

  Adds additional shorthand methods for all common HTTP verbs, a `post_form()` method for POST-ing x-www-form-urlencoded data and a `www_form_urlencode()` utility method.

- IO has been upgraded from version 1.25_04 to 1.25_06, and IO::Handle from version 1.31 to 1.33.

  Together, these upgrades fix a problem with IO::Handle's `getline` and `getlines` methods. When these methods are called on the special ARGV handle, the next file is automatically opened, as happens with the built-in `<>` and `readline` functions. But, unlike the built-ins, these methods were not respecting the caller's use of the open pragma and applying the appropriate I/O layers to the newly-opened file [rt.cpan.org #66474].

- IPC::Cmd has been upgraded from version 0.70 to 0.76.

  Capturing of command output (both `STDOUT` and `STDERR`) is now supported using IPC::Open3 on MSWin32 without requiring IPC::Run.

- IPC::Open3 has been upgraded from version 1.09 to 1.12.

  Fixes a bug which prevented use of `open3` on Windows when `*STDIN`, `*STDOUT` or `*STDERR` had been localized.

  Fixes a bug which prevented duplicating numeric file descriptors on Windows.

  `open3` with "-" for the program name works once more. This was broken in version 1.06 (and hence in Perl 5.14.0) [perl #95748].

- Locale::Codes has been upgraded from version 3.16 to 3.21.

  Added Language Extension codes (langext) and Language Variation codes (langvar) as defined in the IANA language registry.

  Added language codes from ISO 639-5

  Added language/script codes from the IANA language subtag registry

  Fixed an uninitialized value warning [rt.cpan.org #67438].

  Fixed the return value for the all_XXX_codes and all_XXX_names functions [rt.cpan.org #69100].

  Reorganized modules to move Locale::MODULE to Locale::Codes::MODULE to allow for cleaner future additions. The original four modules (Locale::Language, Locale::Currency, Locale::Country, Locale::Script) will continue to work, but all new sets of codes will be added in the Locale::Codes namespace.

  The code2XXX, XXX2code, all_XXX_codes, and all_XXX_names functions now support retired codes. All codesets may be specified by a constant or by their name now. Previously, they were specified only by a constant.

The alias_code function exists for backward compatibility. It has been replaced by rename_country_code. The alias_code function will be removed some time after September, 2013.

All work is now done in the central module (Locale::Codes). Previously, some was still done in the wrapper modules (Locale::Codes::*). Added Language Family codes (langfam) as defined in ISO 639-5.

- Math::BigFloat has been upgraded from version 1.993 to 1.997.

  The `numify` method has been corrected to return a normalized Perl number (the result of `0 + $thing`), instead of a string [rt.cpan.org #66732].

- Math::BigInt has been upgraded from version 1.994 to 1.998.

  It provides a new `bsgn` method that complements the `babs` method.

  It fixes the internal `objectify` function's handling of ‘‘foreign objects’’ so they are converted to the appropriate class (Math::BigInt or Math::BigFloat).

- Math::BigRat has been upgraded from version 0.2602 to 0.2603.

  `int()` on a Math::BigRat object containing -1/2 now creates a Math::BigInt containing 0, rather than -0. Math::BigInt does not even support negative zero, so the resulting object was actually malformed [perl #95530].

- Math::Complex has been upgraded from version 1.56 to 1.59 and Math::Trig from version 1.2 to 1.22.

  Fixes include: correct copy constructor usage; fix polarwise formatting with numeric format specifier; and more stable `great_circle_direction` algorithm.

- Module::CoreList has been upgraded from version 2.51 to 2.66.

  The `corelist` utility now understands the `-r` option for displaying Perl release dates and the `--diff` option to print the set of modlib changes between two perl distributions.

- Module::Metadata has been upgraded from version 1.000004 to 1.000009.

  Adds `provides` method to generate a CPAN META provides data structure correctly; use of `package_versions_from_directory` is discouraged.

- ODBM_File has been upgraded from version 1.10 to 1.12.

  The XS code is now compiled with `PERL_NO_GET_CONTEXT`, which will aid performance under ithreads.

- open has been upgraded from version 1.08 to 1.10.

  It no longer turns off layers on standard handles when invoked without the ‘‘:std’’ directive. Similarly, when invoked *with* the ‘‘:std’’ directive, it now clears layers on STDERR before applying the new ones, and not just on STDIN and STDOUT [perl #92728].

- overload has been upgraded from version 1.13 to 1.18.

  `overload::Overloaded` no longer calls `can` on the class, but uses another means to determine whether the object has overloading. It was never correct for it to call `can`, as overloading does not respect AUTOLOAD. So classes that autoload methods and implement`can` no longer ha ve to account for overloading [perl #40333].

  A warning is now produced for invalid arguments. See ‘‘New Diagnostics’’.

- PerlIO::scalar has been upgraded from version 0.11 to 0.14.

  (This is the module that implements `open $fh, '>', \$scalar`.)

  It fixes a problem with `open my $fh, ">", \$scalar` not working if `$scalar` is a copy-on-write scalar. (5.14.2)

It also fixes a hang that occurs with `readline` or `<$fh>` if a typeglob has been assigned to `$scalar` [perl #92258].

It no longer assumes during `seek` that `$scalar` is a string internally. If it didn't crash, it was close to doing so [perl #92706]. Also, the internal print routine no longer assumes that the position set by `seek` is valid, but extends the string to that position, filling the intervening bytes (between the old length and the seek position) with nulls [perl #78980].

Printing to an in-memory handle now works if the `$scalar` holds a reference, stringifying the reference before modifying it. References used to be treated as empty strings.

Printing to an in-memory handle no longer crashes if the `$scalar` happens to hold a number internally, but no string buffer.

Printing to an in-memory handle no longer creates scalars that confuse the regular expression engine [perl #108398].

- Pod::Functions has been upgraded from version 1.04 to 1.05.

  Functions.pm is now generated at perl build time from annotations in *perlfunc.pod*. This will ensure that Pod::Functions and perlfunc(1) remain in synchronisation.

- Pod::Html has been upgraded from version 1.11 to 1.1502.

  This is an extensive rewrite of Pod::Html to use Pod::Simple under the hood. The output has changed significantly.

- Pod::Perldoc has been upgraded from version 3.15_03 to 3.17.

  It corrects the search paths on VMS [perl #90640]. (5.14.1)

  The **-v** option now fetches the right section for `$0`.

  This upgrade has numerous significant fixes. Consult its changelog on the CPAN for more information.

- POSIX has been upgraded from version 1.24 to 1.30.

  POSIX no longer uses AutoLoader. Any code which was relying on this implementation detail was buggy, and may fail because of this change. The module's Perl code has been considerably simplified, roughly halving the number of lines, with no change in functionality. The XS code has been refactored to reduce the size of the shared object by about 12%, with no change in functionality. More POSIX functions now have tests.

  `sigsuspend` and `pause` now run signal handlers before returning, as the whole point of these two functions is to wait until a signal has arrived, and then return *after* it has been triggered. Delayed, or "safe", signals were preventing that from happening, possibly resulting in race conditions [perl #107216].

  `POSIX::sleep` is now a direct call into the underlying OS `sleep` function, instead of being a Perl wrapper on `CORE::sleep` `POSIX::dup2` now returns the correct value on Win32 (*i.e.*, the file descriptor). `POSIX::SigSet` `sigsuspend` and `sigpending` and `POSIX::pause` now dispatch safe signals immediately before returning to their caller.

  `POSIX::Termios::setattr` now defaults the third argument to `TCSANOW`, instead of 0. On most platforms `TCSANOW` is defined to be 0, but on some 0 is not a valid parameter, which caused a call with defaults to fail.

- Socket has been upgraded from version 1.94 to 2.001.

  It has new functions and constants for handling IPv6 sockets:

```
pack_ipv6_mreq
unpack_ipv6_mreq
IPV6_ADD_MEMBERSHIP
IPV6_DROP_MEMBERSHIP
IPV6_MTU
IPV6_MTU_DISCOVER
IPV6_MULTICAST_HOPS
IPV6_MULTICAST_IF
IPV6_MULTICAST_LOOP
IPV6_UNICAST_HOPS
IPV6_V6ONLY
```

- Storable has been upgraded from version 2.27 to 2.34.

  It no longer turns copy-on-write scalars into read-only scalars when freezing and thawing.

- Sys::Syslog has been upgraded from version 0.27 to 0.29.

  This upgrade closes many outstanding bugs.

- Term::ANSIColor has been upgraded from version 3.00 to 3.01.

  Only interpret an initial array reference as a list of colors, not any initial reference, allowing the colored function to work properly on objects with stringification defined.

- Term::ReadLine has been upgraded from version 1.07 to 1.09.

  Term::ReadLine now supports any event loop, including unpublished ones and simple IO::Select, loops without the need to rewrite existing code for any particular framework [perl #108470].

- threads::shared has been upgraded from version 1.37 to 1.40.

  Destructors on shared objects used to be ignored sometimes if the objects were referenced only by shared data structures. This has been mostly fixed, but destructors may still be ignored if the objects still exist at global destruction time [perl #98204].

- Unicode::Collate has been upgraded from version 0.73 to 0.89.

  Updated to CLDR 1.9.1

  Locales updated to CLDR 2.0: mk, mt, nb, nn, ro, ru, sk, sr, sv, uk, zh__pinyin, zh__stroke

  Newly supported locales: bn, fa, ml, mr, or, pa, sa, si, si__dictionary, sr_Latn, sv__reformed, ta, te, th, ur, wae.

  Tailored compatibility ideographs as well as unified ideographs for the locales: ja, ko, zh__big5han, zh__gb2312han, zh__pinyin, zh__stroke.

  Locale/*.pl files are now searched for in `@INC`.

- Unicode::Normalize has been upgraded from version 1.10 to 1.14.

  Fixes for the removal of *unicore/CompositionExclusions.txt* from core.

- Unicode::UCD has been upgraded from version 0.32 to 0.43.

  This adds four new functions: `prop_aliases()` and `prop_value_aliases()`, which are used to find all Unicode-approved synonyms for property names, or to convert from one name to another; `prop_invlist` which returns all code points matching a given Unicode binary property; and `prop_invmap` which returns the complete specification of a given Unicode property.

- Win32API::File has been upgraded from version 0.1101 to 0.1200.

  Added SetStdHandle and GetStdHandle functions

Removed Modules and Pragmata

As promised in Perl 5.14.0's release notes, the following modules have been removed from the core distribution, and if needed should be installed from CPAN instead.

- Devel::DProf has been removed from the Perl core. Prior version was 20110228.00.
- Shell has been removed from the Perl core. Prior version was 0.72_01.
- Several old perl4-style libraries which have been deprecated with 5.14 are now removed:

  ```
  abbrev.pl assert.pl bigfloat.pl bigint.pl bigrat.pl cacheout.pl
  complete.pl ctime.pl dotsh.pl exceptions.pl fastcwd.pl flush.pl
  getcwd.pl getopt.pl getopts.pl hostname.pl importenv.pl
  lib/find{,depth}.pl look.pl newgetopt.pl open2.pl open3.pl
  pwd.pl shellwords.pl stat.pl tainted.pl termcap.pl timelocal.pl
  ```

  They can be found on CPAN as Perl4::CoreLibs.

Documentation

New Documentation

perldtrace(1)

perldtrace(1) describes Perl's DTrace support, listing the provided probes and gives examples of their use.

perlexperiment(1)

This document is intended to provide a list of experimental features in Perl. It is still a work in progress.

perlootut(1)

This a new OO tutorial. It focuses on basic OO concepts, and then recommends that readers choose an OO framework from CPAN.

perlxstypemap(1)

The new manual describes the XS typemapping mechanism in unprecedented detail and combines new documentation with information extracted from perlxs(1) and the previously unofficial list of all core typemaps.

Changes to Existing Documentation

perlapi(1)

- The HV API has long accepted negative lengths to show that the key is in UTF8. This is now documented.
- The `boolSV()` macro is now documented.

perlfunc(1)

- `dbmopen` treats a 0 mode as a special case, that prevents a nonexistent file from being created. This has been the case since Perl 5.000, but was never documented anywhere. Now the perlfunc(1) entry mentions it [perl #90064].
- As an accident of history, `open $fh, '<:', ...` applies the default layers for the platform (`:raw` on Unix, `:crlf` on Windows), ignoring whatever is declared by open.pm. This seems such a useful feature it has been documented in perlfunc(1) and open.
- The entry for `split` has been rewritten. It is now far clearer than before.

perlguts(1)

- A new section, Autoloading with XSUBs, has been added, which explains the two APIs for accessing the name of the autoloaded sub.
- Some function descriptions in perlguts(1) were confusing, as it was not clear whether they referred to the function above or below the description. This has been clarified [perl #91790].

perlobj(1)

- This document has been rewritten from scratch, and its coverage of various OO concepts has been expanded.

perlop(1)

- Documentation of the smartmatch operator has been reworked and moved from perlsyn(1) to perlop(1) where it belongs.

  It has also been corrected for the case of `undef` on the left-hand side. The list of different smart match behaviors had an item in the wrong place.
- Documentation of the ellipsis statement (`...`) has been reworked and moved from perlop(1) to perlsyn.
- The explanation of bitwise operators has been expanded to explain how they work on Unicode strings (5.14.1).
- More examples for `m//g` have been added (5.14.1).
- The `<<\FOO` here-doc syntax has been documented (5.14.1).

perlpragma(1)

- There is now a standard convention for naming keys in the `%^H`, documented under Key naming.

"Laundering and Detecting Tainted Data" in perlsec(1)

- The example function for checking for taintedness contained a subtle error. `$@` needs to be localized to prevent its changing this global's value outside the function. The preferred method to check for this remains "tainted" in Scalar::Util.

perllol(1)

- perllol(1) has been expanded with examples using the new `push $scalar` syntax introduced in Perl 5.14.0 (5.14.1).

perlmod(1)

- perlmod(1) now states explicitly that some types of explicit symbol table manipulation are not supported. This codifies what was effectively already the case [perl #78074].

perlpodstyle(1)

- The tips on which formatting codes to use have been corrected and greatly expanded.
- There are now a couple of example one-liners for previewing POD files after they have been edited.

perlre(1)

- The `(*COMMIT)` directive is now listed in the right section (Verbs without an argument).

perlrun(1)

- perlrun(1) has undergone a significant clean-up. Most notably, the **-0x...** form of the **-0** flag has been clarified, and the final section on environment variables has been corrected and expanded (5.14.1).

perlsub(1)

- The ($;) prototype syntax, which has existed for rather a long time, is now documented in perlsub. It lets a unary function have the same precedence as a list operator.

perltie(1)

- The required syntax for tying handles has been documented.

perlvar(1)

- The documentation for $! has been corrected and clarified. It used to state that $! could be `undef`, which is not the case. It was also unclear whether system calls set C's `errno` or Perl's `$!` [perl #91614].

- Documentation for $$ has been amended with additional cautions regarding changing the process ID.

Other Changes

- perlxs(1) was extended with documentation on inline typemaps.
- perlref(1) has a new Circular References section explaining how circularities may not be freed and how to solve that with weak references.
- Parts of perlapi(1) were clarified, and Perl equivalents of some C functions have been added as an additional mode of exposition.
- A few parts of perlre(1) and perlrecharclass(1) were clarified.

Removed Documentation

Old OO Documentation

The old OO tutorials, perltoot(1), perltooc(1), and perlboot(1), have been removed. The perlbot(1) (bag of object tricks) document has been removed as well.

Development Deltas

The perldelta(1) files for development releases are no longer packaged with perl. These can still be found in the perl source code repository.

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

- Cannot set tied `@DB::args`

  This error occurs when `caller` tries to set `@DB::args` but finds it tied. Before this error was added, it used to crash instead.

- Cannot tie unreifiable array

  This error is part of a safety check that the `tie` operator does before tying a special array like `@_`. You should never see this message.

- &CORE::%s cannot be called directly

  This occurs when a subroutine in the `CORE::` namespace is called with `&foo` syntax or through a reference. Some subroutines in this package cannot yet be called that way, but must be called as barewords. See "Subroutines in the `CORE` namespace", above.

- Source filters apply only to byte streams

  This new error occurs when you try to activate a source filter (usually by loading a source filter module) within a string passed to `eval` under the `unicode_eval` feature.

New Warnings

- defined(@array) is deprecated

  The long-deprecated `defined(@array)` now also warns for package variables. Previously it issued a warning for lexical variables only.

- *length()* used on `%s`

  This new warning occurs when `length` is used on an array or hash, instead of `scalar(@array)` or `scalar(keys %hash)`.

- lvalue attribute `%s` already-defined subroutine

  attributes.pm now emits this warning when the :lvalue attribute is applied to a Perl subroutine that has already been defined, as doing so can have unexpected side-effects.

- overload arg '%s' is invalid

  This warning, in the "overload" category, is produced when the overload pragma is given an argument it doesn't recognize, presumably a mistyped operator.

- $[used in `%s` (did you mean $] ?)

  This new warning exists to catch the mistaken use of `$[` in version checks. `$]`, not `$[`, contains the version number.

- Useless assignment to a temporary

  Assigning to a temporary scalar returned from an lvalue subroutine now produces this warning [perl #31946].

- Useless use of \E

  `\E` does nothing unless preceded by `\Q`, `\L` or `\U`.

Removed Errors

- "sort is now a reserved word"

  This error used to occur when `sort` was called without arguments, followed by `;` or `)`. (E.g., `sort;` would die, but `{sort}` was OK.) This error message was added in Perl 3 to catch code like `close(sort)` which would no longer work. More than two decades later, this message is no longer appropriate. Now `sort` without arguments is always allowed, and returns an empty list, as it did in those cases where it was already allowed [perl #90030].

Changes to Existing Diagnostics

- The "Applying pattern match..." or similar warning produced when an array or hash is on the left-hand side of the `=~` operator now mentions the name of the variable.
- The "Attempt to free non-existent shared string" has had the spelling of "non-existent" corrected to "nonexistent". It was already listed with the correct spelling in perldiag.
- The error messages for using `default` and `when` outside a topicalizer have been standardized to match the messages for `continue` and loop controls. They now read 'Can't "default" outside a topicalizer' and 'Can't "when" outside a topicalizer'. They both used to be 'Can't use *when()* outside a topicalizer' [perl #91514].
- The message, "Code point 0x%X is not Unicode, no properties match it; all inverse properties do" has been changed to "Code point 0x%X is not Unicode, all \p{} matches fail; all \P{} matches succeed".
- Redefinition warnings for constant subroutines used to be mandatory, even occurring under `no warnings`. Now they respect the warnings pragma.
- The "glob failed" warning message is now suppressible via `no warnings` [perl #111656].
- The Invalid version format error message now says "negative version number" within the parentheses, rather than "non-numeric data", for negative numbers.
- The two warnings Possible attempt to put comments in *qw()* list and Possible attempt to separate words with commas are no longer mutually exclusive: the same `qw` construct may produce both.
- The uninitialized warning for `y///r` when `$_` is implicit and undefined now mentions the variable name, just like the non-/r variation of the operator.
- The 'Use of "foo" without parentheses is ambiguous' warning has been extended to apply also to user-defined subroutines with a (;$) prototype, and not just to built-in functions.
- Warnings that mention the names of lexical (`my`) variables with Unicode characters in them now respect the presence or absence of the `:utf8` layer on the output handle, instead of outputting UTF8 regardless. Also, the correct names are included in the strings passed to `$SIG{__WARN__}` handlers, rather than the raw UTF8 bytes.

Utility Changes

h2ph

- h2ph used to generate code of the form

  ```
  unless(defined(&FOO)) {
  sub FOO () {42;}
  }
  ```

  But the subroutine is a compile-time declaration, and is hence unaffected by the condition. It has now been corrected to emit a string `eval` around the subroutine [perl #99368].

splain

- *splain* no longer emits backtraces with the first line number repeated.

  This:

  ```
  Uncaught exception from user code:
  Cannot fwiddle the fwuddle at -e line 1.
  at -e line 1
  main::baz() called at -e line 1
  main::bar() called at -e line 1
  main::foo() called at -e line 1
  ```

  has become this:

  ```
  Uncaught exception from user code:
  Cannot fwiddle the fwuddle at -e line 1.
  main::baz() called at -e line 1
  main::bar() called at -e line 1
  main::foo() called at -e line 1
  ```

- Some error messages consist of multiple lines that are listed as separate entries in perldiag. splain has been taught to find the separate entries in these cases, instead of simply failing to find the message.

zipdetails

- This is a new utility, included as part of an IO::Compress::Base upgrade.

  zipdetails displays information about the internal record structure of the zip file. It is not concerned with displaying any details of the compressed data stored in the zip file.

Configuration and Compilation

- *regexp.h* has been modified for compatibility with GCC's **-Werror** option, as used by some projects that include perl's header files (5.14.1).
- `USE_LOCALE{,_COLLATE,_CTYPE,_NUMERIC}` have been added the output of perl -V as they have affect the behavior of the interpreter binary (albeit in only a small area).
- The code and tests for IPC::Open2 have been moved from *ext/IPC-Open2* into *ext/IPC-Open3*, as `IPC::Open2::open2()` is implemented as a thin wrapper around `IPC::Open3::_open3()`, and hence is very tightly coupled to it.
- The magic types and magic vtables are now generated from data in a new script *regen/mg_vtable.pl*, instead of being maintained by hand. As different EBCDIC variants can't agree on the code point for '~', the character to code point conversion is done at build time by *generate_uudmap* to a new generated header *mg_data.h*. `PL_vtbl_bm` and `PL_vtbl_fm` are now defined by the pre-processor as `PL_vtbl_regexp`, instead of being distinct C variables. `PL_vtbl_sig` has been removed.
- Building with `-DPERL_GLOBAL_STRUCT` works again. This configuration is not generally used.
- Perl configured with *MAD* now correctly frees `MADPROP` structures when OPs are freed. `MADPROP`s are now allocated with `PerlMemShared_malloc()`

- *makedef.pl* has been refactored. This should have no noticeable affect on any of the platforms that use it as part of their build (AIX, VMS, Win32).
- `useperlio` can no longer be disabled.
- The file *global.sym* is no longer needed, and has been removed. It contained a list of all exported functions, one of the files generated by *regen/embed.pl* from data in *embed.fnc* and *regen/opcodes*. The code has been refactored so that the only user of *global.sym*, *makedef.pl*, now reads *embed.fnc* and *regen/opcodes* directly, removing the need to store the list of exported functions in an intermediate file.

  As *global.sym* was never installed, this change should not be visible outside the build process.
- *pod/buildtoc*, used by the build process to build perltoc(1), has been refactored and simplified. It now contains only code to build perltoc; the code to regenerate Makefiles has been moved to *Porting/pod_rules.pl*. It's a bug if this change has any material effect on the build process.
- *pod/roffitall* is now built by *pod/buildtoc*, instead of being shipped with the distribution. Its list of manpages is now generated (and therefore current). See also RT #103202 for an unresolved related issue.
- The man page for `XS::Typemap` is no longer installed. `XS::Typemap` is a test module which is not installed, hence installing its documentation makes no sense.
- The -Dusesitecustomize and -Duserelocatableinc options now work together properly.

Platform Support

Platform-Specific Notes

Cygwin

- Since version 1.7, Cygwin supports native UTF-8 paths. If Perl is built under that environment, directory and filenames will be UTF-8 encoded.
- Cygwin does not initialize all original Win32 environment variables. See *README.cygwin* for a discussion of the newly-added `Cygwin::sync_winenv()` function [perl #110190] and for further links.

HP-UX

- HP-UX PA-RISC/64 now supports gcc-4.x

  A fix to correct the socketsize now makes the test suite pass on HP-UX PA-RISC for 64bitall builds. (5.14.2)

VMS

- Remove unnecessary includes, fix miscellaneous compiler warnings and close some unclosed comments on *vms/vms.c*.
- Remove sockadapt layer from the VMS build.
- Explicit support for VMS versions before v7.0 and DEC C versions before v6.0 has been removed.
- Since Perl 5.10.1, the home-grown `stat` wrapper has been unable to distinguish between a directory name containing an underscore and an otherwise-identical filename containing a dot in the same position (e.g., t/test_pl as a directory and t/test.pl as a file). This problem has been corrected.
- The build on VMS now permits names of the resulting symbols in C code for Perl longer than 31 characters. Symbols like `Perl__it_was_the_best_of_times_it_was_the_worst_of_times` can now be created freely without causing the VMS linker to seize up.

GNU/Hurd

- Numerous build and test failures on GNU/Hurd have been resolved with hints for building DBM modules, detection of the library search path, and enabling of large file support.

OpenVOS

- Perl is now built with dynamic linking on OpenVOS, the minimum supported version of which is now Release 17.1.0.

SunOS

The CC workshop C++ compiler is now detected and used on systems that ship without cc.

Internal Changes

- The compiled representation of formats is now stored via the `mg_ptr` of their `PERL_MAGIC_fm`. Previously it was stored in the string buffer, beyond `SvLEN()`, the regular end of the string. `SvCOMPILED()` and `SvCOMPILED_{on,off}()` now exist solely for compatibility for XS code. The first is always 0, the other two now no-ops. (5.14.1)
- Some global variables have been marked `const`, members in the interpreter structure have been re-ordered, and the opcodes have been re-ordered. The op `OP_AELEMFAST` has been split into `OP_AELEMFAST` and `OP_AELEMFAST_LEX`.
- When empting a hash of its elements (e.g., via undef(%h), or `%h`=()), HvARRAY field is no longer temporarily zeroed. Any destructors called on the freed elements see the remaining elements. Thus, `%h`=() becomes more like `delete $h{$_} for keys %h`.
- Boyer-Moore compiled scalars are now PVMGs, and the Boyer-Moore tables are now stored via the mg_ptr of their `PERL_MAGIC_bm`. Previously they were PVGVs, with the tables stored in the string buffer, beyond `SvLEN()`. This eliminates the last place where the core stores data beyond `SvLEN()`.
- Simplified logic in `Perl_sv_magic()` introduces a small change of behavior for error cases involving unknown magic types. Previously, if `Perl_sv_magic()` was passed a magic type unknown to it, it would

  1. Croak ''Modification of a read-only value attempted'' if read only
  2. Return without error if the SV happened to already have this magic
  3. otherwise croak ''Don't know how to handle magic of type \\%o''

  Now it will always croak ''Don't know how to handle magic of type \\%o'', even on read-only values, or SVs which already have the unknown magic type.
- The experimental `fetch_cop_label` function has been renamed to `cop_fetch_label`.
- The `cop_store_label` function has been added to the API, but is experimental.
- *embedvar.h* has been simplified, and one level of macro indirection for PL_* variables has been removed for the default (non-multiplicity) configuration. PERLVAR*() macros now directly expand their arguments to tokens such as `PL_defgv`, instead of expanding to `PL_Idefgv`, with *embedvar.h* defining a macro to map `PL_Idefgv` to `PL_defgv`. XS code which has unwarranted chumminess with the implementation may need updating.
- An API has been added to explicitly choose whether to export XSUB symbols. More detail can be found in the comments for commit e64345f8.
- The `is_gv_magical_sv` function has been eliminated and merged with `gv_fetchpvn_flags`. It used to be called to determine whether a GV should be autovivified in rvalue context. Now it has been replaced with a new `GV_ADDMG` flag (not part of the API).
- The returned code point from the function `utf8n_to_uvuni()` when the input is malformed UTF-8, malformations are allowed, and `utf8` warnings are off is now the Unicode REPLACEMENT CHARACTER whenever the malformation is such that no well-defined code point can be computed. Previously the returned value was essentially garbage. The only malformations that have well-defined values are a zero-length string (0 is the return), and overlong UTF-8 sequences.
- Padlists are now marked `AvREAL`; i.e., reference-counted. They have always been reference-counted, but were not marked real, because *pad.c* did its own clean-up, instead of using the usual clean-up code in *sv.c*. That caused problems in thread cloning, so now the `AvREAL` flag is on, but is turned off in *pad.c* right before the padlist is freed (after *pad.c* has done its custom freeing of the pads).

- All C files that make up the Perl core have been converted to UTF-8.
- These new functions have been added as part of the work on Unicode symbols:

  ```
  HvNAMELEN
  HvNAMEUTF8
  HvENAMELEN
  HvENAMEUTF8
  gv_init_pv
  gv_init_pvn
  gv_init_pvsv
  gv_fetchmeth_pv
  gv_fetchmeth_pvn
  gv_fetchmeth_sv
  gv_fetchmeth_pv_autoload
  gv_fetchmeth_pvn_autoload
  gv_fetchmeth_sv_autoload
  gv_fetchmethod_pv_flags
  gv_fetchmethod_pvn_flags
  gv_fetchmethod_sv_flags
  gv_autoload_pv
  gv_autoload_pvn
  gv_autoload_sv
  newGVgen_flags
  sv_derived_from_pv
  sv_derived_from_pvn
  sv_derived_from_sv
  sv_does_pv
  sv_does_pvn
  sv_does_sv
  whichsig_pv
  whichsig_pvn
  whichsig_sv
  newCONSTSUB_flags
  ```

  The gv_fetchmethod_*_flags functions, like gv_fetchmethod_flags, are experimental and may change in a future release.
- The following functions were added. These are *not* part of the API:

  ```
  GvNAMEUTF8
  GvENAMELEN
  GvENAME_HEK
  CopSTASH_flags
  CopSTASH_flags_set
  PmopSTASH_flags
  PmopSTASH_flags_set
  sv_sethek
  HEKfARG
  ```

  There is also a `HEKf` macro corresponding to `SVf`, for interpolating HEKs in formatted strings.
- `sv_catpvn_flags` takes a couple of new internal-only flags, `SV_CATBYTES` and `SV_CATUTF8`, which tell it whether the char array to be concatenated is UTF8. This allows for more efficient concatenation than creating temporary SVs to pass to `sv_catsv`.
- For XS AUTOLOAD subs, `$AUTOLOAD` is set once more, as it was in 5.6.0. This is in addition to setting `SvPVX(cv)`, for compatibility with 5.8 to 5.14. See ''Autoloading with XSUBs'' in perlguts.

- Perl now checks whether the array (the linearized isa) returned by a MRO plugin begins with the name of the class itself, for which the array was created, instead of assuming that it does. This prevents the first element from being skipped during method lookup. It also means that `mro::get_linear_isa` may return an array with one more element than the MRO plugin provided [perl #94306].
- `PL_curstash` is now reference-counted.
- There are now feature bundle hints in `PL_hints` (`$^H`) that version declarations use, to avoid having to load *feature.pm*. One setting of the hint bits indicates a "custom" feature bundle, which means that the entries in `%^H` still apply. *feature.pm* uses that.

  The `HINT_FEATURE_MASK` macro is defined in *perl.h* along with other hints. Other macros for setting and testing features and bundles are in the new *feature.h*. `FEATURE_IS_ENABLED` (which has moved to *feature.h*) is no longer used throughout the codebase, but more specific macros, e.g., `FEATURE_SAY_IS_ENABLED`, that are defined in *feature.h*.
- *lib/feature.pm* is now a generated file, created by the new *regen/feature.pl* script, which also generates *feature.h*.
- Tied arrays are now always `AvREAL`. If `@_` or `DB::args` is tied, it is reified first, to make sure this is always the case.
- Two new functions `utf8_to_uvchr_buf()` and `utf8_to_uvuni_buf()` have been added. These are the same as `utf8_to_uvchr` and `utf8_to_uvuni` (which are now deprecated), but take an extra parameter that is used to guard against reading beyond the end of the input string. See "utf8_to_uvchr_buf" in perlapi(1) and "utf8_to_uvuni_buf" in perlapi.
- The regular expression engine now does TRIE case insensitive matches under Unicode. This may change the output of `use re 'debug';`, and will speed up various things.
- There is a new `wrap_op_checker()` function, which provides a thread-safe alternative to writing to `PL_check` directly.

Selected Bug Fixes

Array and hash

- A bug has been fixed that would cause a "Use of freed value in iteration" error if the next two hash elements that would be iterated over are deleted [perl #85026]. (5.14.1)
- Deleting the current hash iterator (the hash element that would be returned by the next call to `each`) in void context used not to free it [perl #85026].
- Deletion of methods via `delete $Class::{method}` syntax used to update method caches if called in void context, but not scalar or list context.
- When hash elements are deleted in void context, the internal hash entry is now freed before the value is freed, to prevent destructors called by that latter freeing from seeing the hash in an inconsistent state. It was possible to cause double-frees if the destructor freed the hash itself [perl #100340].
- A `keys` optimization in Perl 5.12.0 to make it faster on empty hashes caused `each` not to reset the iterator if called after the last element was deleted.
- Freeing deeply nested hashes no longer crashes [perl #44225].
- It is possible from XS code to create hashes with elements that have no values. The hash element and slice operators used to crash when handling these in lvalue context. They now produce a "Modification of non-creatable hash value attempted" error message.
- If list assignment to a hash or array triggered destructors that freed the hash or array itself, a crash would ensue. This is no longer the case [perl #107440].
- It used to be possible to free the typeglob of a localized array or hash (e.g., `local @{"x"}; delete $::{x}`), resulting in a crash on scope exit.

- Some core bugs affecting Hash::Util have been fixed: locking a hash element that is a glob copy no longer causes the next assignment to it to corrupt the glob (5.14.2), and unlocking a hash element that holds a copy-on-write scalar no longer causes modifications to that scalar to modify other scalars that were sharing the same string buffer.

C API fixes

- The `newHVhv` XS function now works on tied hashes, instead of crashing or returning an empty hash.
- The `SvIsCOW` C macro now returns false for read-only copies of typeglobs, such as those created by:

  ```
  $hash{elem} = *foo;
  Hash::Util::lock_value %hash, 'elem';
  ```

  It used to return true.
- The `SvPVutf8` C function no longer tries to modify its argument, resulting in errors [perl #108994].
- `SvPVutf8` now works properly with magical variables.
- `SvPVbyte` now works properly non-PVs.
- When presented with malformed UTF-8 input, the XS-callable functions `is_utf8_string()`, `is_utf8_string_loc()`, and `is_utf8_string_loclen()` could read beyond the end of the input string by up to 12 bytes. This no longer happens. [perl #32080]. However, currently, `is_utf8_char()` still has this defect, see "*is_utf8_char()*" above.
- The C-level `pregcomp` function could become confused about whether the pattern was in UTF8 if the pattern was an overloaded, tied, or otherwise magical scalar [perl #101940].

Compile-time hints

- Tying `%^H` no longer causes perl to crash or ignore the contents of `%^H` when entering a compilation scope [perl #106282].
- `eval $string`and `require` used not to localize `%^H` during compilation if it w as empty at the time the `eval` call itself was compiled. This could lead to scary side effects, like `use re "/m"` enabling other flags that the surrounding code was trying to enable for its caller [perl #68750].
- `eval $string`and `require` no longer localize hints (`$^H` and `%^H`) at run time, b ut only during compilation of the `$string` or required file. This makes `BEGIN { $^H{foo}=7 }` equivalent to `BEGIN { eval '$^H{foo}=7' }`[perl #70151].
- Creating a BEGIN block from XS code (via `newXS` or `newATTRSUB`) would, on completion, make the hints of the current compiling code the current hints. This could cause warnings to occur in a non-warning scope.

Copy-on-write scalars

Copy-on-write or shared hash key scalars were introduced in 5.8.0, but most Perl code did not encounter them (they were used mostly internally). Perl 5.10.0 extended them, such that assigning `__PACKAGE__` or a hash key to a scalar would make it copy-on-write. Several parts of Perl were not updated to account for them, but have now been fixed.

- `utf8::decode` had a nasty bug that would modify copy-on-write scalars' string buffers in place (i.e., skipping the copy). This could result in hashes having two elements with the same key [perl #91834]. (5.14.2)
- Lvalue subroutines were not allowing COW scalars to be returned. This was fixed for lvalue scalar context in Perl 5.12.3 and 5.14.0, but list context was not fixed until this release.
- Elements of restricted hashes (see the fields pragma) containing copy-on-write values couldn't be deleted, nor could such hashes be cleared (`%hash = ()`). (5.14.2)
- Localizing a tied variable used to make it read-only if it contained a copy-on-write string. (5.14.2)
- Assigning a copy-on-write string to a stash element no longer causes a double free. Regardless of this change, the results of such assignments are still undefined.

- Assigning a copy-on-write string to a tied variable no longer stops that variable from being tied if it happens to be a PVMG or PVLV internally.
- Doing a substitution on a tied variable returning a copy-on-write scalar used to cause an assertion failure or an ‘‘Attempt to free nonexistent shared string’’ warning.
- This one is a regression from 5.12: In 5.14.0, the bitwise assignment operators `|=`, `^=` and `&=` started leaving the left-hand side undefined if it happened to be a copy-on-write string [perl #108480].
- Storable, Devel::Peek and PerlIO::scalar had similar problems. See ‘‘Updated Modules and Pragmata’’, above.

The debugger

- *dumpvar.pl*, and therefore the `x` command in the debugger, have been fixed to handle objects blessed into classes whose names contain ‘‘=’’. The contents of such objects used not to be dumped [perl #101814].
- The ‘‘R’’ command for restarting a debugger session has been fixed to work on Windows, or any other system lacking a `POSIX::_SC_OPEN_MAX` constant [perl #87740].
- The `#line 42 foo` directive used not to update the arrays of lines used by the debugger if it occurred in a string eval. This was partially fixed in 5.14, but it worked only for a single `#line 42 foo` in each eval. Now it works for multiple.
- When subroutine calls are intercepted by the debugger, the name of the subroutine or a reference to it is stored in `$DB::sub`, for the debugger to access. Sometimes (such as `$foo = *bar; undef *bar; &$foo`) `$DB::sub` would be set to a name that could not be used to find the subroutine, and so the debugger's attempt to call it would fail. Now the check to see whether a reference is needed is more robust, so those problems should not happen anymore [rt.cpan.org #69862].
- Every subroutine has a filename associated with it that the debugger uses. The one associated with constant subroutines used to be misallocated when cloned under threads. Consequently, debugging threaded applications could result in memory corruption [perl #96126].

Dereferencing operators

- `defined(${"..."})`, `defined(*{"..."})`, etc., used to return true for most, but not all built-in variables, if they had not been used yet. This bug affected `${^GLOBAL_PHASE}` and `${^UTF8CACHE}`, among others. It also used to return false if the package name was given as well (`${"::!"}`) [perl #97978, #97492].
- Perl 5.10.0 introduced a similar bug: `defined(*{"foo"})` where ‘‘foo’’ represents the name of a built-in global variable used to return false if the variable had never been used before, but only on the *first* call. This, too, has been fixed.
- Since 5.6.0, `*{ ... }` has been inconsistent in how it treats undefined values. It would die in strict mode or lvalue context for most undefined values, but would be treated as the empty string (with a warning) for the specific scalar return by `undef()` (`&PL_sv_undef` internally). This has been corrected. `undef()` is now treated like other undefined scalars, as in Perl 5.005.

Filehandle, last-accessed

Perl has an internal variable that stores the last filehandle to be accessed. It is used by `$.` and by `tell` and `eof` without arguments.

- It used to be possible to set this internal variable to a glob copy and then modify that glob copy to be something other than a glob, and still have the last-accessed filehandle associated with the variable after assigning a glob to it again:

```
my $foo = *STDOUT; # $foo is a glob copy
<$foo>; # $foo is now the last-accessed handle
$foo = 3; # no longer a glob
$foo = *STDERR; # still the last-accessed handle
```

  Now the `$foo = 3` assignment unsets that internal variable, so there is no last-accessed filehandle, just as if `<$foo>` had never happened.

This also prevents some unrelated handle from becoming the last-accessed handle if `$foo` falls out of scope and the same internal SV gets used for another handle [perl #97988].

- A regression in 5.14 caused these statements not to set that internal variable:

  ```
  my $fh = *STDOUT;
  tell $fh;
  eof $fh;
  seek $fh, 0,0;
  tell *$fh;
  eof *$fh;
  seek *$fh, 0,0;
  readline *$fh;
  ```

  This is now fixed, but `tell *{ *$fh }` still has the problem, and it is not clear how to fix it [perl #106536].

Filetests and `stat`

The term ‘‘filetests’’ refers to the operators that consist of a hyphen followed by a single letter: `-r`, `-x`, `-M`, etc. The term ‘‘stacked’’ when applied to filetests means followed by another filetest operator sharing the same operand, as in `-r -x -w $fooo`.

- `stat` produces more consistent warnings. It no longer warns for ‘‘_’’ [perl #71002] and no longer skips the warning at times for other unopened handles. It no longer warns about an unopened handle when the operating system's `fstat` function fails.
- `stat` would sometimes return negative numbers for large inode numbers, because it was using the wrong internal C type. [perl #84590]
- `lstat` is documented to fall back to `stat` (with a warning) when given a filehandle. When passed an IO reference, it was actually doing the equivalent of `stat_` and ignoring the handle.
- `-T _`with no preceding `stat` used to produce a confusing ‘ ‘uninitialized’’ warning, even though there is no visible uninitialized value to speak of.
- `-T`, `-B`, `-l` and `-t` now work when stacked with other filetest operators [perl #77388].
- In 5.14.0, filetest ops (`-r`, `-x`, etc.) started calling FETCH on a tied argument belonging to the previous argument to a list operator, if called with a bareword argument or no argument at all. This has been fixed, so `push @foo, $tied, -r` no longer calls FETCH on `$tied`.
- In Perl 5.6, `-l` followed by anything other than a bareword would treat its argument as a file name. That was changed in 5.8 for glob references (`\*foo`), but not for globs themselves (`*foo`). `-l` started returning `undef` for glob references without setting the last stat buffer that the ‘‘_’’ handle uses, but only if warnings were turned on. With warnings off, it was the same as 5.6. In other words, it was simply buggy and inconsistent. Now the 5.6 behavior has been restored.
- `-l` followed by a bareword no longer ‘‘eats’’ the previous argument to the list operator in whose argument list it resides. Hence, `print "bar", -l foo` now actually prints ‘‘bar’’, because `-l` on longer eats it.
- Perl keeps several internal variables to keep track of the last stat buffer, from which file(handle) it originated, what type it was, and whether the last stat succeeded.

  There were various cases where these could get out of synch, resulting in inconsistent or erratic behavior in edge cases (every mention of `-T` applies to `-B` as well):

  - `-T` *`HANDLE`*, even though it does a `stat`, was not resetting the last stat type, so an `lstat _` following it would merrily return the wrong results. Also, it was not setting the success status.
  - Freeing the handle last used by `stat` or a filetest could result in `-T_` using an unrelated handle.

- `stat` with an IO reference would not reset the stat type or record the filehandle for `-T_` to use.
- Fatal warnings could cause the stat buffer not to be reset for a filetest operator on an unopened filehandle or `-l` on any handle. Fatal warnings also stopped `-T` from setting `$!`.
- When the last stat was on an unreadable file, `-T _` is supposed to return `undef`, leaving the last stat buffer unchanged. But it was setting the stat type, causing `lstat _` to stop working.
- `-T` *`FILENAME`* was not resetting the internal stat buffers for unreadable files.

These have all been fixed.

Formats

- Several edge cases have been fixed with formats and `formline`; in particular, where the format itself is potentially variable (such as with ties and overloading), and where the format and data differ in their encoding. In both these cases, it used to possible for the output to be corrupted [perl #91032].
- `formline` no longer converts its argument into a string in-place. So passing a reference to `formline` no longer destroys the reference [perl #79532].
- Assignment to `$^A` (the format output accumulator) now recalculates the number of lines output.

`given` **and** `when`

- `given` was not scoping its implicit `$_` properly, resulting in memory leaks or ‘‘Variable is not available’’ warnings [perl #94682].
- `given` was not calling set-magic on the implicit lexical `$_` that it uses. This meant, for example, that `pos` would be remembered from one execution of the same `given` block to the next, even if the input were a different variable [perl #84526].
- `when` blocks are now capable of returning variables declared inside the enclosing `given` block [perl #93548].

The `glob` **operator**

- On OSes other than VMS, Perl’s `glob` operator (and the `<...>` form) use File::Glob underneath. File::Glob splits the pattern into words, before feeding each word to its `bsd_glob` function.

  There were several inconsistencies in the way the split was done. Now quotation marks (’ and ") are always treated as shell-style word delimiters (that allow whitespace as part of a word) and backslashes are always preserved, unless they exist to escape quotation marks. Before, those would only sometimes be the case, depending on whether the pattern contained whitespace. Also, escaped whitespace at the end of the pattern is no longer stripped [perl #40470].
- `CORE::glob` now works as a way to call the default globbing function. It used to respect overrides, despite the `CORE::` prefix.
- Under miniperl (used to configure modules when perl itself is built), `glob` now clears `%ENV` before calling csh, since the latter croaks on some systems if it does not like the contents of the LS_COLORS environment variable [perl #98662].

Lvalue subroutines

- Explicit return now returns the actual argument passed to return, instead of copying it [perl #72724, #72706].
- Lvalue subroutines used to enforce lvalue syntax (i.e., whatever can go on the left-hand side of `=`) for the last statement and the arguments to return. Since lvalue subroutines are not always called in lvalue context, this restriction has been lifted.
- Lvalue subroutines are less restrictive about what values can be returned. It used to croak on values returned by `shift` and `delete` and from other subroutines, but no longer does so [perl #71172].
- Empty lvalue subroutines (`sub :lvalue {}`) used to return `@_` in list context. All subroutines used to do this, but regular subs were fixed in Perl 5.8.2. Now lvalue subroutines have been likewise fixed.
- Autovivification now works on values returned from lvalue subroutines [perl #7946], as does returning `keys` in lvalue context.

- Lvalue subroutines used to copy their return values in rvalue context. Not only was this a waste of CPU cycles, but it also caused bugs. A `($)` prototype would cause an lvalue sub to copy its return value [perl #51408], and `while(lvalue_sub() =~ m/.../g) { ... }` would loop endlessly [perl #78680].
- When called in potential lvalue context (e.g., subroutine arguments or a list passed to `for`), lvalue subroutines used to copy any read-only value that was returned. E.g., `sub :lvalue { $] }` would not return `$]`, but a copy of it.
- When called in potential lvalue context, an lvalue subroutine returning arrays or hashes used to bind the arrays or hashes to scalar variables, resulting in bugs. This was fixed in 5.14.0 if an array were the first thing returned from the subroutine (but not for `$scalar, @array` or hashes being returned). Now a more general fix has been applied [perl #23790].
- Method calls whose arguments were all surrounded with `my()` or `our()` (as in `$object->method(my($a,$b))`) used to force lvalue context on the subroutine. This would prevent lvalue methods from returning certain values.
- Lvalue sub calls that are not determined to be such at compile time (`&$name` or &{"name"}) are no longer exempt from strict refs if they occur in the last statement of an lvalue subroutine [perl #102486].
- Sub calls whose subs are not visible at compile time, if they occurred in the last statement of an lvalue subroutine, would reject non-lvalue subroutines and die with "Can't modify non-lvalue subroutine call" [perl #102486].

  Non-lvalue sub calls whose subs *are* visible at compile time exhibited the opposite bug. If the call occurred in the last statement of an lvalue subroutine, there would be no error when the lvalue sub was called in lvalue context. Perl would blindly assign to the temporary value returned by the non-lvalue subroutine.
- `AUTOLOAD` routines used to take precedence over the actual sub being called (i.e., when autoloading wasn't needed), for sub calls in lvalue or potential lvalue context, if the subroutine was not visible at compile time.
- Applying the `:lvalue` attribute to an XSUB or to an aliased subroutine stub with `sub foo :lvalue;` syntax stopped working in Perl 5.12. This has been fixed.
- Applying the :lvalue attribute to subroutine that is already defined does not work properly, as the attribute changes the way the sub is compiled. Hence, Perl 5.12 began warning when an attempt is made to apply the attribute to an already defined sub. In such cases, the attribute is discarded.

  But the change in 5.12 missed the case where custom attributes are also present: that case still silently and ineffectively applied the attribute. That omission has now been corrected. `sub foo :lvalue :Whatever` (when `foo` is already defined) now warns about the :lvalue attribute, and does not apply it.
- A bug affecting lvalue context propagation through nested lvalue subroutine calls has been fixed. Previously, returning a value in nested rvalue context would be treated as lvalue context by the inner subroutine call, resulting in some values (such as read-only values) being rejected.

Overloading

- Arithmetic assignment (`$left += $right`) involving overloaded objects that rely on the 'nomethod' override no longer segfault when the left operand is not overloaded.
- Errors that occur when methods cannot be found during overloading now mention the correct package name, as they did in 5.8.x, instead of erroneously mentioning the "overload" package, as they have since 5.10.0.
- Undefining `%overload::` no longer causes a crash.

Prototypes of built-in keywords

- The `prototype` function no longer dies for the `__FILE__`, `__LINE__` and `__PACKAGE__` directives. It now returns an empty-string prototype for them, because they are syntactically indistinguishable from nullary functions like `time`.
- `prototype` now returns `undef` for all overridable infix operators, such as `eq`, which are not callable in any way resembling functions. It used to return incorrect prototypes for some and die for others [perl #94984].
- The prototypes of several built-in functions—`getprotobynumber`, `lock`, `not` and `select`--have been corrected, or at least are now closer to reality than before.

Regular expressions

- `/[[:ascii:]]/` and `/[[:blank:]]/` now use locale rules under `use locale` when the platform supports that. Previously, they used the platform's native character set.
- `m/[[:ascii:]]/i` and `/\p{ASCII}/i` now match identically (when not under a differing locale). This fixes a regression introduced in 5.14 in which the first expression could match characters outside of ASCII, such as the KELVIN SIGN.
- `/.*/g` would sometimes refuse to match at the end of a string that ends with "\n". This has been fixed [perl #109206].
- Starting with 5.12.0, Perl used to get its internal bookkeeping muddled up after assigning `${ qr// }` to a hash element and locking it with Hash::Util. This could result in double frees, crashes, or erratic behavior.
- The new (in 5.14.0) regular expression modifier `/a` when repeated like `/aa` forbids the characters outside the ASCII range that match characters inside that range from matching under `/i`. This did not work under some circumstances, all involving alternation, such as:

  ```
   "\N{KELVIN SIGN}" =~ /k|foo/iaa;
  ```

  succeeded inappropriately. This is now fixed.
- 5.14.0 introduced some memory leaks in regular expression character classes such as `[\w\s]`, which have now been fixed. (5.14.1)
- An edge case in regular expression matching could potentially loop. This happened only under `/i` in bracketed character classes that have characters with multi-character folds, and the target string to match against includes the first portion of the fold, followed by another character that has a multi-character fold that begins with the remaining portion of the fold, plus some more.

  ```
   "s\N{U+DF}" =~ /[\x{DF}foo]/i
  ```

  is one such case. `\xDF` folds to `"ss"`. (5.14.1)
- A few characters in regular expression pattern matches did not match correctly in some circumstances, all involving `/i`. The affected characters are: COMBINING GREEK YPOGEGRAMMENI, GREEK CAPITAL LETTER IOTA, GREEK CAPITAL LETTER UPSILON, GREEK PROSGEGRAMMENI, GREEK SMALL LETTER IOTA WITH DIALYTIKA AND OXIA, GREEK SMALL LETTER IOTA WITH DIALYTIKA AND TONOS, GREEK SMALL LETTER UPSILON WITH DIALYTIKA AND OXIA, GREEK SMALL LETTER UPSILON WITH DIALYTIKA AND TONOS, LATIN SMALL LETTER LONG S, LATIN SMALL LIGATURE LONG S T, and LATIN SMALL LIGATURE ST.
- A memory leak regression in regular expression compilation under threading has been fixed.
- A regression introduced in 5.14.0 has been fixed. This involved an inverted bracketed character class in a regular expression that consisted solely of a Unicode property. That property wasn't getting inverted outside the Latin1 range.
- Three problematic Unicode characters now work better in regex pattern matching under `/i`.

  In the past, three Unicode characters: LATIN SMALL LETTER SHARP S, GREEK SMALL LETTER IOTA WITH DIALYTIKA AND TONOS, and GREEK SMALL LETTER UPSILON WITH DIALYTIKA AND

TONOS, along with the sequences that they fold to (including "ss" for LATIN SMALL LETTER SHARP S), did not properly match under `/i`. 5.14.0 fixed some of these cases, but introduced others, including a panic when one of the characters or sequences was used in the `(?(DEFINE)` regular expression predicate. The known bugs that were introduced in 5.14 have now been fixed; as well as some other edge cases that have never worked until now. These all involve using the characters and sequences outside bracketed character classes under `/i`. This closes [perl #98546].

There remain known problems when using certain characters with multi-character folds inside bracketed character classes, including such constructs as `qr/[\N{LATIN SMALL LETTER SHARP}a-z]/i`. These remaining bugs are addressed in [perl #89774].

- RT #78266: The regex engine has been leaking memory when accessing named captures that weren't matched as part of a regex ever since 5.10 when they were introduced; e.g., this would consume over a hundred MB of memory:

```
for (1..10_000_000) {
if ("foo" =~ /(foo|(?<capture>bar))?/) {
my $capture = $+{capture}
}
}
system "ps -o rss $$"'
```

- In 5.14, `/[[:lower:]]/i` and `/[[:upper:]]/i` no longer matched the opposite case. This has been fixed [perl #101970].
- A regular expression match with an overloaded object on the right-hand side would sometimes stringify the object too many times.
- A regression has been fixed that was introduced in 5.14, in `/i` regular expression matching, in which a match improperly fails if the pattern is in UTF-8, the target string is not, and a Latin-1 character precedes a character in the string that should match the pattern. [perl #101710]
- In case-insensitive regular expression pattern matching, no longer on UTF-8 encoded strings does the scan for the start of match look only at the first possible position. This caused matches such as `"f\x{FB00}" =~ /ff/i`to f ail.
- The regexp optimizer no longer crashes on debugging builds when merging fixed-string nodes with inconvenient contents.
- A panic involving the combination of the regular expression modifiers `/aa` and the `\b` escape sequence introduced in 5.14.0 has been fixed [perl #95964]. (5.14.2)
- The combination of the regular expression modifiers `/aa` and the `\b` and `\B` escape sequences did not work properly on UTF-8 encoded strings. All non-ASCII characters under `/aa` should be treated as non-word characters, but what was happening was that Unicode rules were used to determine wordness/non-wordness for non-ASCII characters. This is now fixed [perl #95968].
- `(?foo: ...)`no longer loses passed in character set.
- The trie optimization used to have problems with alternations containing an empty `(?:)`, causing `"x" =~ /\A(?>(?:(?:)A|B|C?x))\z/`not to match, whereas it should [perl #111842].
- Use of lexical (`my`) variables in code blocks embedded in regular expressions will no longer result in memory corruption or crashes.

  Nevertheless, these code blocks are still experimental, as there are still problems with the wrong variables being closed over (in loops for instance) and with abnormal exiting (e.g., `die`) causing memory corruption.
- The `\h`, `\H`, `\v` and `\V` regular expression metacharacters used to cause a panic error message when trying to match at the end of the string [perl #96354].
- The abbreviations for four C1 control characters `MW` `PM`, `RI`, and `ST` were previously unrecognized by `\N{}`, *vianame()*, and *string_vianame()*.

- Mentioning a variable named “&” other than `$&` (i.e., `@&` or `%&`) no longer stops `$&` from working. The same applies to variables named “’” and “‘” [perl #24237].
- Creating a `UNIVERSAL::AUTOLOAD` sub no longer stops `%+`, `%-` and `%!` from working some of the time [perl #105024].

Smartmatching

- `~~` now correctly handles the precedence of Any~~Object, and is not tricked by an overloaded object on the left-hand side.
- In Perl 5.14.0, `$tainted ~~ @array` stopped working properly. Sometimes it would erroneously fail (when `$tainted` contained a string that occurs in the array *after* the first element) or erroneously succeed (when `undef` occurred after the first element) [perl #93590].

The `sort` operator

- `sort` was not treating `sub {}` and `sub {()}` as equivalent when such a sub was provided as the comparison routine. It used to croak on `sub {()}`.
- `sort` now works once more with custom sort routines that are XSUBs. It stopped working in 5.10.0.
- `sort` with a constant for a custom sort routine, although it produces unsorted results, no longer crashes. It started crashing in 5.10.0.
- Warnings emitted by `sort` when a custom comparison routine returns a non-numeric value now contain “in sort” and show the line number of the `sort` operator, rather than the last line of the comparison routine. The warnings also now occur only if warnings are enabled in the scope where `sort` occurs. Previously the warnings would occur if enabled in the comparison routine’s scope.
- `sort { $a <=> $b }`, which is optimized internally, now produces “uninitialized” warnings for NaNs (not-a-number values), since `<=>` returns `undef` for those. This brings it in line with `sort{1;$a<=>$b}` and other more complex cases, which are not optimized [perl #94390].

The `substr` operator

- Tied (and otherwise magical) variables are no longer exempt from the “Attempt to use reference as lvalue in substr” warning.
- That warning now occurs when the returned lvalue is assigned to, not when `substr` itself is called. This makes a difference only if the return value of `substr` is referenced and later assigned to.
- Passing a substring of a read-only value or a typeglob to a function (potential lvalue context) no longer causes an immediate “Can’t coerce” or “Modification of a read-only value” error. That error occurs only if the passed value is assigned to.

  The same thing happens with the “substr outside of string” error. If the lvalue is only read from, not written to, it is now just a warning, as with rvalue `substr`.
- `substr` assignments no longer call FETCH twice if the first argument is a tied variable, just once.

Support for embedded nulls

Some parts of Perl did not work correctly with nulls (`chr 0`) embedded in strings. That meant that, for instance, `$m = "a\0b"; foo->$m` would call the “a” method, instead of the actual method name contained in `$m`. These parts of perl have been fixed to support nulls:

- Method names
- Typeglob names (including filehandle and subroutine names)
- Package names, including the return value of `ref()`
- Typeglob elements (`*foo{"THING\0stuff"}`)
- Signal names
- Various warnings and error messages that mention variable names or values, methods, etc.

One side effect of these changes is that blessing into “\0” no longer causes `ref()` to return false.

Threading bugs

- Typeglobs returned from threads are no longer cloned if the parent thread already has a glob with the same name. This means that returned subroutines will now assign to the right package variables [perl #107366].
- Some cases of threads crashing due to memory allocation during cloning have been fixed [perl #90006].
- Thread joining would sometimes emit "Attempt to free unreferenced scalar" warnings if `caller` had been used from the `DB` package before thread creation [perl #98092].
- Locking a subroutine (via `lock &sub`) is no longer a compile-time error for regular subs. For lvalue subroutines, it no longer tries to return the sub as a scalar, resulting in strange side effects like `ref \$_` returning "CODE" in some instances.

  `lock &sub`is no w a run-time error if threads::shared is loaded (a no-op otherwise), but that may be rectified in a future version.

Tied variables

- Various cases in which FETCH was being ignored or called too many times have been fixed:
  - `PerlIO::get_layers` [perl #97956]
  - `$tied =~ y/a/b/`, `chop $tied` and `chomp $tied` when `$tied` holds a reference.
  - When calling `local $_` [perl #105912]
  - Four-argument `select`
  - A tied buffer passed to `sysread`
  - `$tied .= <>`
  - Three-argument `open`, the third being a tied file handle (as in `open $fh, ">&", $tied`)
  - `sort` with a reference to a tied glob for the comparison routine.
  - `..` and `...` in list context [perl #53554].
  - `${$tied}`, `@{$tied}`, `%{$tied}` and `*{$tied}` where the tied variable returns a string (`&{}` was unaffected)
  - `defined ${ $tied_variable }`
  - Various functions that take a filehandle argument in rvalue context (`close`, `readline`, etc.) [perl #97482]
  - Some cases of dereferencing a complex expression, such as `${ (), $tied } = 1`, used to call `FETCH` multiple times, but now call it once.
  - `$tied->method` where `$tied` returns a package name—even resulting in a failure to call the method, due to memory corruption
  - Assignments like `*$tied = \&{"..."}` and `*glob = $tied`
  - `chdir`, `chmod`, `chown`, `utime`, `truncate`, `stat`, `lstat` and the filetest ops (`-r`, `-x`, etc.)
- `caller` sets `@DB::args` to the subroutine arguments when called from the DB package. It used to crash when doing so if `@DB::args` happened to be tied. Now it croaks instead.
- Tying an element of `%ENV` or `%^H` and then deleting that element would result in a call to the tie object's DELETE method, even though tying the element itself is supposed to be equivalent to tying a scalar (the element is, of course, a scalar) [perl #67490].
- When Perl autovivifies an element of a tied array or hash (which entails calling STORE with a new reference), it now calls FETCH immediately after the STORE, instead of assuming that FETCH would have returned the same reference. This can make it easier to implement tied objects [perl #35865, #43011].

- Four-argument `select` no longer produces its "Non-string passed as bitmask" warning on tied or tainted variables that are strings.
- Localizing a tied scalar that returns a typeglob no longer stops it from being tied till the end of the scope.
- Attempting to `goto` out of a tied handle method used to cause memory corruption or crashes. Now it produces an error message instead [perl #8611].
- A bug has been fixed that occurs when a tied variable is used as a subroutine reference: if the last thing assigned to or returned from the variable was a reference or typeglob, the `\&$tied` could either crash or return the wrong subroutine. The reference case is a regression introduced in Perl 5.10.0. For typeglobs, it has probably never worked till now.

Version objects and vstrings

- The bitwise complement operator (and possibly other operators, too) when passed a vstring would leave vstring magic attached to the return value, even though the string had changed. This meant that `version->new(~v1.2.3)` would create a version looking like "v1.2.3" even though the string passed to `version->new` was actually "\376\375\374". This also caused B::Deparse to deparse `~v1.2.3` incorrectly, without the ~ [perl #29070].
- Assigning a vstring to a magic (e.g., tied, `$!`) variable and then assigning something else used to blow away all magic. This meant that tied variables would come undone, `$!` would stop getting updated on failed system calls, `$|` would stop setting autoflush, and other mischief would take place. This has been fixed.
- `version->new("version")` and `printf "%vd", "version"` no longer crash [perl #102586].
- Version comparisons, such as those that happen implicitly with `use v5.43`, no longer cause locale settings to change [perl #105784].
- Version objects no longer cause memory leaks in boolean context [perl #109762].

Warnings, redefinition

- Subroutines from the `autouse` namespace are once more exempt from redefinition warnings. This used to work in 5.005, but was broken in 5.6 for most subroutines. For subs created via XS that redefine subroutines from the `autouse` package, this stopped working in 5.10.
- New XSUBs now produce redefinition warnings if they overwrite existing subs, as they did in 5.8.x. (The `autouse` logic was reversed in 5.10-14. Only subroutines from the `autouse` namespace would warn when clobbered.)
- `newCONSTSUB` used to use compile-time warning hints, instead of run-time hints. The following code should never produce a redefinition warning, but it used to, if `newCONSTSUB` redefined an existing subroutine:

  ```
  use warnings;
  BEGIN {
  no warnings;
  some_XS_function_that_calls_new_CONSTSUB();
  }
  ```
- Redefinition warnings for constant subroutines are on by default (what are known as severe warnings in perldiag). This occurred only when it was a glob assignment or declaration of a Perl subroutine that caused the warning. If the creation of XSUBs triggered the warning, it was not a default warning. This has been corrected.
- The internal check to see whether a redefinition warning should occur used to emit "uninitialized" warnings in cases like this:

```
use warnings "uninitialized";
use constant {u => undef, v => undef};
sub foo(){u}
sub foo(){v}
```

Warnings, "Uninitialized"

- Various functions that take a filehandle argument in rvalue context (`close`, `readline`, etc.) used to warn twice for an undefined handle [perl #97482].
- `dbmopen` now only warns once, rather than three times, if the mode argument is `undef` [perl #90064].
- The `+=` operator does not usually warn when the left-hand side is `undef`, but it was doing so for tied variables. This has been fixed [perl #44895].
- A bug fix in Perl 5.14 introduced a new bug, causing "uninitialized" warnings to report the wrong variable if the operator in question had two operands and one was `%{...}` or `@{...}`. This has been fixed [perl #103766].
- `..` and `...` in list context now mention the name of the variable in "uninitialized" warnings for string (as opposed to numeric) ranges.

Weak references

- Weakening the first argument to an automatically-invoked `DESTROY` method could result in erroneous "DESTROY created new reference" errors or crashes. Now it is an error to weaken a read-only reference.
- Weak references to lexical hashes going out of scope were not going stale (becoming undefined), but continued to point to the hash.
- Weak references to lexical variables going out of scope are now broken before any magical methods (e.g., DESTROY on a tie object) are called. This prevents such methods from modifying the variable that will be seen the next time the scope is entered.
- Creating a weak reference to an `@ISA` array or accessing the array index (`$#ISA`) could result in confused internal bookkeeping for elements later added to the `@ISA` array. For instance, creating a weak reference to the element itself could push that weak reference on to `@ISA`; and elements added after use of `$#ISA` would be ignored by method lookup [perl #85670].

Other notable fixes

- `quotemeta` now quotes consistently the same non-ASCII characters under `use feature 'unicode_strings'`, regardless of whether the string is encoded in UTF-8 or not, hence fixing the last vestiges (we hope) of the notorious "The "Unicode Bug"" in perlunicode. [perl #77654].

  Which of these code points is quoted has changed, based on Unicode's recommendations. See "quotemeta" in perlfunc(1) for details.
- `study` is now a no-op, presumably fixing all outstanding bugs related to study causing regex matches to behave incorrectly!
- When one writes `open foo || die`, which used to work in Perl 4, a "Precedence problem" warning is produced. This warning used erroneously to apply to fully-qualified bareword handle names not followed by `||`. This has been corrected.
- After package aliasing (`*foo:: = *bar::`), `select` with 0 or 1 argument would sometimes return a name that could not be used to refer to the filehandle, or sometimes it would return `undef` even when a filehandle was selected. Now it returns a typeglob reference in such cases.
- `PerlIO::get_layers` no longer ignores some arguments that it thinks are numeric, while treating others as filehandle names. It is now consistent for flat scalars (i.e., not references).
- Unrecognized switches on `#!` line

  If a switch, such as **-x**, that cannot occur on the `#!` line is used there, perl dies with "Can't emulate...''.

It used to produce the same message for switches that perl did not recognize at all, whether on the command line or the `#!` line.

Now it produces the ``Unrecognized switch'' error message [perl #104288].

- `system` now temporarily blocks the SIGCHLD signal handler, to prevent the signal handler from stealing the exit status [perl #105700].
- The `%n` formatting code for `printf` and `sprintf`, which causes the number of characters to be assigned to the next argument, now actually assigns the number of characters, instead of the number of bytes.

  It also works now with special lvalue functions like `substr` and with nonexistent hash and array elements [perl #3471, #103492].
- Perl skips copying values returned from a subroutine, for the sake of speed, if doing so would make no observable difference. Because of faulty logic, this would happen with the result of `delete`, `shift` or `splice`, even if the result was referenced elsewhere. It also did so with tied variables about to be freed [perl #91844, #95548].
- `utf8::decode` now refuses to modify read-only scalars [perl #91850].
- Freeing `$_` inside a `grep` or `map` block, a code block embedded in a regular expression, or an `@INC` filter (a subroutine returned by a subroutine in `@INC`) used to result in double frees or crashes [perl #91880, #92254, #92256].
- `eval` returns `undef` in scalar context or an empty list in list context when there is a run-time error. When `eval` was passed a string in list context and a syntax error occurred, it used to return a list containing a single undefined element. Now it returns an empty list in list context for all errors [perl #80630].
- `goto &func`no longer crashes, b ut produces an error message, when the unwinding of the current subroutine's scope fires a destructor that undefines the subroutine being ``goneto'' [perl #99850].
- Perl now holds an extra reference count on the package that code is currently compiling in. This means that the following code no longer crashes [perl #101486]:

  ```
  package Foo;
  BEGIN {*Foo:: = *Bar::}
  sub foo;
  ```
- The `x` repetition operator no longer crashes on 64-bit builds with large repeat counts [perl #94560].
- Calling `require` on an implicit `$_` when `*CORE::GLOBAL::require` has been overridden does not segfault anymore, and `$_` is now passed to the overriding subroutine [perl #78260].
- `use` and `require` are no longer affected by the I/O layers active in the caller's scope (enabled by open.pm) [perl #96008].
- `our $::é; $é`(which is in valid) no longer produces the ``Compilation error at lib/utf8_heavy.pl...'' error message, which it started emitting in 5.10.0 [perl #99984].
- On 64-bit systems, `read()` now understands large string offsets beyond the 32-bit range.
- Errors that occur when processing subroutine attributes no longer cause the subroutine's op tree to leak.
- Passing the same constant subroutine to both `index` and `formline` no longer causes one or the other to fail [perl #89218]. (5.14.1)
- List assignment to lexical variables declared with attributes in the same statement (`my ($x,@y) : blimp = (72,94)`) stopped working in Perl 5.8.0. It has now been fixed.
- Perl 5.10.0 introduced some faulty logic that made ``U*'' in the middle of a pack template equivalent to ``U0'' if the input string was empty. This has been fixed [perl #90160]. (5.14.2)

- Destructors on objects were not called during global destruction on objects that were not referenced by any scalars. This could happen if an array element were blessed (e.g., `bless \$a[0]`) or if a closure referenced a blessed variable (`bless \my @a; sub foo { @a }`).

  Now there is an extra pass during global destruction to fire destructors on any objects that might be left after the usual passes that check for objects referenced by scalars [perl #36347].
- Fixed a case where it was possible that a freed buffer may have been read from when parsing a here document [perl #90128]. (5.14.1)
- `each(`*`ARRAY`*`)` is now wrapped in `defined(...)`, like `each(`*`HASH`*`)`, inside a `while` condition [perl #90888].
- A problem with context propagation when a `do` block is an argument to `return` has been fixed. It used to cause `undef` to be returned in certain cases of a `return` inside an `if` block which itself is followed by another `return`.
- Calling `index` with a tainted constant no longer causes constants in subsequently compiled code to become tainted [perl #64804].
- Infinite loops like `1 while 1` used to stop `strict 'subs'` mode from working for the rest of the block.
- For list assignments like `($a,$b) = ($b,$a)`, Perl has to make a copy of the items on the right-hand side before assignment them to the left. For efficiency's sake, it assigns the values on the right straight to the items on the left if no one variable is mentioned on both sides, as in `($a,$b) = ($c,$d)`. The logic for determining when it can cheat was faulty, in that `&&` and `||` on the right-hand side could fool it. So `($a,$b) = $some_true_value && ($b,$a)` would end up assigning the value of `$b` to both scalars.
- Perl no longer tries to apply lvalue context to the string in `("string", $variable) ||= 1` (which used to be an error). Since the left-hand side of `||=` is evaluated in scalar context, that's a scalar comma operator, which gives all but the last item void context. There is no such thing as void lvalue context, so it was a mistake for Perl to try to force it [perl #96942].
- `caller` no longer leaks memory when called from the DB package if `@DB::args` was assigned to after the first call to `caller`. Carp was triggering this bug [perl #97010]. (5.14.2)
- `close` and similar filehandle functions, when called on built-in global variables (like `$+`), used to die if the variable happened to hold the undefined value, instead of producing the usual ``Use of uninitialized value'' warning.
- When autovivified file handles were introduced in Perl 5.6.0, `readline` was inadvertently made to autovivify when called as `readline($foo)` (but not as `<$foo>`). It has now been fixed never to autovivify.
- Calling an undefined anonymous subroutine (e.g., what `$x` holds after `undef &{$x = sub{}}`) used to cause a ``Not a CODE reference'' error, which has been corrected to ``Undefined subroutine called'' [perl #71154].
- Causing `@DB::args` to be freed between uses of `caller` no longer results in a crash [perl #93320].
- `setpgrp($foo)` used to be equivalent to `($foo, setpgrp)`, because `setpgrp` was ignoring its argument if there was just one. Now it is equivalent to `setpgrp($foo,0)`.
- `shmread` was not setting the scalar flags correctly when reading from shared memory, causing the existing cached numeric representation in the scalar to persist [perl #98480].
- `++` and `--` now work on copies of globs, instead of dying.
- `splice()` doesn't warn when truncating

  You can now limit the size of an array using `splice(@a,MAX_LEN)` without worrying about warnings.

- `$$` is no longer tainted. Since this value comes directly from `getpid()`, it is always safe.
- The parser no longer leaks a filehandle if STDIN was closed before parsing started [perl #37033].
- `die;` with a non-reference, non-string, or magical (e.g., tainted) value in $@ now properly propagates that value [perl #111654].

Known Problems

- On Solaris, we have two kinds of failure.

  If *make* is Sun's *make,* we get an error about a badly formed macro assignment in the *Makefile*. That happens when *./Configure* tries to make depends. *Configure* then exits 0, but further *make*-ing fails.

  If *make* is *gmake*, *Configure* completes, then we get errors related to */usr/include/stdbool.h*
- On Win32, a number of tests hang unless STDERR is redirected. The cause of this is still under investigation.
- When building as root with a umask that prevents files from being other-readable, *t/op/filetest.t* will fail. This is a test bug, not a bug in perl's behavior.
- Configuring with a recent gcc and link-time-optimization, such as `Configure -Doptimize='-O2 -flto'`f ails because the optimizer optimizes away some of Configure's tests. A workaround is to omit the `-flto` flag when running Configure, but add it back in while actually building, something like

  ```
  sh Configure -Doptimize=-O2
  make OPTIMIZE='-O2 -flto'
  ```
- The following CPAN modules have test failures with perl 5.16. Patches have been submitted for all of these, so hopefully there will be new releases soon:
  - Date::Pcalc version 6.1
  - Module::CPANTS::Analyse version 0.85

    This fails due to problems in Module::Find 0.10 and File::MMagic 1.27.
  - PerlIO::Util version 0.72

Acknowledgements

Perl 5.16.0 represents approximately 12 months of development since Perl 5.14.0 and contains approximately 590,000 lines of changes across 2,500 files from 139 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 5.16.0:

Aaron Crane, Abhijit Menon-Sen, Abigail, Alan Haggai Alavi, Alberto Simões, Alexandr Ciornii, Andreas König, Andy Dougherty, Aristotle Pagaltzis, Bo Johansson, Bo Lindbergh, Breno G. de Oliveira, brian d foy, Brian Fraser, Brian Greenfield, Carl Hayter, Chas. Owens, Chia-liang Kao, Chip Salzenberg, Chris 'BinGOs' Williams, Christian Hansen, Christopher J. Madsen, chromatic, Claes Jacobsson, Claudio Ramirez, Craig A. Berry, Damian Conway, Daniel Kahn Gillmor, Darin McBride, Dave Rolsky, David Cantrell, David Golden, David Leadbeater, David Mitchell, Dee Newcum, Dennis Kaarsemaker, Dominic Hargreaves, Douglas Christopher Wilson, Eric Brine, Father Chrysostomos, Florian Ragwitz, Frederic Briere, George Greer, Gerard Goossen, Gisle Aas, H.Merijn Brand, Hojung Youn, Ian Goodacre, James E Keenan, Jan Dubois, Jerry D. Hedden, Jesse Luehrs, Jesse Vincent, Jilles Tjoelker, Jim Cromie, Jim Meyering, Joel Berger, Johan Vromans, Johannes Plunien, John Hawkinson, John P. Linderman, John Peacock, Joshua ben Jore, Juerd Waalboer, Karl Williamson, Karthik Rajagopalan, Keith Thompson, Kevin J. Woolley, Kevin Ryde, Laurent Dami, Leo Lapworth, Leon Brocard, Leon Timmermans, Louis Strous, Lukas Mai, Marc Green, Marcel Grünauer, Mark A. Stratman, Mark Dootson, Mark Jason Dominus, Martin Hasch, Matthew Horsfall, Max Maischein, Michael G Schwern, Michael Witten, Mike Sheldrake, Moritz Lenz, Nicholas Clark, Niko Tyni, Nuno Carvalho, Pau Amma, Paul Evans, Paul Green, Paul Johnson, Perlover, Peter John Acklam, Peter Martini, Peter Scott, Phil Monsen, Pino Toscano, Rafael Garcia-Suarez, Rainer Tammer, Reini Urban, Ricardo Signes, Robin Barker, Rodolfo Carvalho, Salvador

Fandiño, Sam Kimbrel, Samuel Thibault, Shawn M Moore, Shigeya Suzuki, Shirakata Kentaro, Shlomi Fish, Sisyphus, Slaven Rezic, Spiros Denaxas, Steffen Müller, Steffen Schwigon, Stephen Bennett, Stephen Oberholtzer, Stevan Little, Steve Hay, Steve Peters, Thomas Sibley, Thorsten Glaser, Timothe Litt, Todd Rinaldo, Tom Christiansen, Tom Hukins, Tony Cook, Vadim Konovalov, Vincent Pit, Vladimir Timofeev, Walt Mankowski, Yves Orton, Zefram, Zsbán Ambrus, Ævar Arnfjörð Bjarmason.

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

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

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

Reporting Bugs

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

If you believe you have an unreported bug, please run the perlbug(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 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 use this address only 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.