NAME

perl5100delta - what is new for perl 5.10.0

DESCRIPTION

This document describes the differences between the 5.8.8 release and the 5.10.0 release.

Many of the bug fixes in 5.10.0 were already seen in the 5.8.X maintenance releases; they are not duplicated here and are documented in the set of man pages named perl58[1-8]?delta.

Core Enhancements

The `feature` pragma

The `feature` pragma is used to enable new syntax that would break Perl's backwards-compatibility with older releases of the language. It's a lexical pragma, like `strict` or `warnings`.

Currently the following new features are available: `switch` (adds a switch statement), `say` (adds a `say` built-in function), and `state` (adds a `state` keyword for declaring ''static'' variables). Those features are described in their own sections of this document.

The `feature` pragma is also implicitly loaded when you require a minimal perl version (with the `use VERSION` construct) greater than, or equal to, 5.9.5. See feature for details.

New −E command-line switch

-E is equivalent to **-e**, but it implicitly enables all optional features (like `use feature ":5.10"`).

Defined-or operator

A new operator `//` (defined-or) has been implemented. The following expression:

```
$a // $b
```

is merely equivalent to

```
defined $a ? $a : $b
```

and the statement

```
$c //= $d;
```

can now be used instead of

```
$c = $d unless defined $c;
```

The `//` operator has the same precedence and associativity as `||`. Special care has been taken to ensure that this operator Do What You Mean while not breaking old code, but some edge cases involving the empty regular expression may now parse differently. See perlop(1) for details.

Switch and Smart Match operator

Perl 5 now has a switch statement. It's available when `use feature 'switch'` is in effect. This feature introduces three new keywords, `given`, `when`, and `default`:

```
given ($foo) {
when (/^abc/) { $abc = 1; }
when (/^def/) { $def = 1; }
when (/^xyz/) { $xyz = 1; }
default { $nothing = 1; }
}
```

A more complete description of how Perl matches the switch variable against the `when` conditions is given in ''Switch statements'' in perlsyn.

This kind of match is called *smart match*, and it's also possible to use it outside of switch statements, via the new `~~` operator. See ''Smart matching in detail'' in perlsyn.

This feature was contributed by Robin Houston.

Regular expressions

Recursive Patterns

It is now possible to write recursive patterns without using the `(??{})` construct. This new way is more efficient, and in many cases easier to read.

Each capturing parenthesis can now be treated as an independent pattern that can be entered by using the `(?PARNO)` syntax (`PARNO` standing for ‘‘parenthesis number’’). For example, the following pattern will match nested balanced angle brackets:

```
/
^ # start of line
( # start capture buffer 1
< # match an opening angle bracket
(?: # match one of:
(?> # don't backtrack over the inside of this group
[^<>]+ # one or more non angle brackets
) # end non backtracking group
| # ... or ...
(?1) # recurse to bracket 1 and try it again
)* # 0 or more times.
> # match a closing angle bracket
) # end capture buffer one
$ # end of line
/x
```

PCRE users should note that Perl’s recursive regex feature allows backtracking into a recursed pattern, whereas in PCRE the recursion is atomic or ‘‘possessive’’ in nature. As in the example above, you can add (?>) to control this selectively. (Yves Orton)

Named Capture Buffers

It is now possible to name capturing parenthesis in a pattern and refer to the captured contents by name. The naming syntax is `(?<NAME>....)`. It’s possible to backreference to a named buffer with the `\k<NAME>` syntax. In code, the new magical hashes `%+` and `%-` can be used to access the contents of the capture buffers.

Thus, to replace all doubled chars with a single copy, one could write

```
s/(?<letter>.)\k<letter>/$+{letter}/g
```

Only buffers with defined contents will be ‘‘visible’’ in the `%+` hash, so it’s possible to do something like

```
foreach my $name (keys %+) {
print "content of buffer '$name' is $+{$name}\n";
}
```

The `%-` hash is a bit more complete, since it will contain array refs holding values from all capture buffers similarly named, if there should be many of them.

`%+` and `%-` are implemented as tied hashes through the new module `Tie::Hash::NamedCapture`

Users exposed to the .NET regex engine will find that the perl implementation differs in that the numerical ordering of the buffers is sequential, and not ‘‘unnamed first, then named’’. Thus in the pattern

```
/(A)(?<B>B)(C)(?<D>D)/
```

`$1` will be ’A’, `$2` will be ’B’, `$3` will be ’C’ and `$4` will be ’D’ and not `$1` is ’A’, `$2` is ’C’ and `$3` is ’B’ and `$4` is ’D’ that a .NET programmer would expect. This is considered a feature. :-) (Yves Orton)

Possessive Quantifiers
: Perl now supports the ‘‘possessive quantifier’’ syntax of the ‘‘atomic match’’ pattern. Basically a possessive quantifier matches as much as it can and never gives any back. Thus it can be used to control backtracking. The syntax is similar to non-greedy matching, except instead of using a ’?’ as the modifier the ’+’ is used. Thus `?+`, `*+`, `++`, `{min,max}+` are now legal quantifiers. (Yves Orton)

Backtracking control verbs
: The regex engine now supports a number of special-purpose backtrack control verbs: (*THEN), (*PRUNE), (*MARK), (*SKIP), (*COMMIT), (*FAIL) and (*ACCEPT). See perlre(1) for their descriptions. (Yves Orton)

Relative backreferences
: A new syntax `\g{N}` or `\gN` where ‘‘N’’ is a decimal integer allows a safer form of back-reference notation as well as allowing relative backreferences. This should make it easier to generate and embed patterns that contain backreferences. See ‘‘Capture buffers’’ in perlre. (Yves Orton)

`\K` escape
: The functionality of Jeff Pinyan’s module Regexp::Keep has been added to the core. In regular expressions you can now use the special escape `\K` as a way to do something like floating length positive lookbehind. It is also useful in substitutions like:

```
s/(foo)bar/$1/g
```

that can now be converted to

```
s/foo\Kbar//g
```

which is much more efficient. (Yves Orton)

Vertical and horizontal whitespace, and linebreak
: Regular expressions now recognize the `\v` and `\h` escapes that match vertical and horizontal whitespace, respectively. `\V` and `\H` logically match their complements.

`\R` matches a generic linebreak, that is, vertical whitespace, plus the multi-character sequence `"\x0D\x0A"`.

Optional pre-match and post-match captures with the /p flag
: There is a new flag `/p` for regular expressions. Using this makes the engine preserve a copy of the part of the matched string before the matching substring to the new special variable `${^PREMATCH}`, the part after the matching substring to `${^POSTMATCH}`, and the matched substring itself to `${^MATCH}`.

Perl is still able to store these substrings to the special variables `$``, `$'`, `$&`, but using these variables anywhere in the program adds a penalty to all regular expression matches, whereas if you use the `/p` flag and the new special variables instead, you pay only for the regular expressions where the flag is used.

For more detail on the new variables, see perlvar; for the use of the regular expression flag, see perlop(1) and perlre.

`say()`
: *say()* is a new built-in, only available when `use feature 'say'` is in effect, that is similar to *print()*, but that implicitly appends a newline to the printed string. See ‘‘say’’ in perlfunc. (Robin Houston)

Lexical `$_`
: The default variable `$_` can now be lexicalized, by declaring it like any other lexical variable, with a simple

```
my $_;
```

The operations that default on `$_` will use the lexically-scoped version of `$_` when it exists, instead of the global `$_`.

In a `map` or a `grep` block, if `$_` was previously my’ed, then the `$_` inside the block is lexical as well (and scoped to the block).

In a scope where $_ has been lexicalized, you can still have access to the global version of $_ by using $::_, or, more simply, by overriding the lexical declaration with our $_. (Rafael Garcia-Suarez)

The _ prototype

A new prototype character has been added. _ is equivalent to $ but defaults to $_ if the corresponding argument isn't supplied (both $ and _ denote a scalar). Due to the optional nature of the argument, you can only use it at the end of a prototype, or before a semicolon.

This has a small incompatible consequence: the *prototype()* function has been adjusted to return _ for some built-ins in appropriate cases (for example, prototype('CORE::rmdir')). (Rafael Garcia-Suarez)

UNITCHECK blocks

UNITCHECK, a new special code block has been introduced, in addition to BEGIN, CHECK, INIT and END.

CHECK and INIT blocks, while useful for some specialized purposes, are always executed at the transition between the compilation and the execution of the main program, and thus are useless whenever code is loaded at runtime. On the other hand, UNITCHECK blocks are executed just after the unit which defined them has been compiled. See perlmod(1) for more information. (Alex Gough)

New Pragma, mro

A new pragma, mro (for Method Resolution Order) has been added. It permits to switch, on a per-class basis, the algorithm that perl uses to find inherited methods in case of a multiple inheritance hierarchy. The default MRO hasn't changed (DFS, for Depth First Search). Another MRO is available: the C3 algorithm. See mro for more information. (Brandon Black)

Note that, due to changes in the implementation of class hierarchy search, code that used to undef the *ISA glob will most probably break. Anyway, undef'ing *ISA had the side-effect of removing the magic on the @ISA array and should not have been done in the first place. Also, the cache *::ISA::CACHE:: no longer exists; to force reset the @ISA cache, you now need to use the mro API, or more simply to assign to @ISA (e.g. with @ISA = @ISA).

readdir() **may return a "short filename" on Windows**

The *readdir()* function may return a "short filename" when the long filename contains characters outside the ANSI codepage. Similarly *Cwd::cwd()* may return a short directory name, and *glob()* may return short names as well. On the NTFS file system these short names can always be represented in the ANSI codepage. This will not be true for all other file system drivers; e.g. the FAT filesystem stores short filenames in the OEM codepage, so some files on FAT volumes remain unaccessible through the ANSI APIs.

Similarly, $^X, @INC, and $ENV{PATH} are preprocessed at startup to make sure all paths are valid in the ANSI codepage (if possible).

The *Win32::GetLongPathName()* function now returns the UTF-8 encoded correct long file name instead of using replacement characters to force the name into the ANSI codepage. The new *Win32::GetANSIPathName()* function can be used to turn a long pathname into a short one only if the long one cannot be represented in the ANSI codepage.

Many other functions in the Win32 module have been improved to accept UTF-8 encoded arguments. Please see Win32 for details.

readpipe() **is now overridable**

The built-in function *readpipe()* is now overridable. Overriding it permits also to override its operator counterpart, qx// (a.k.a. ``). Moreover, it now defaults to $_ if no argument is provided. (Rafael Garcia-Suarez)

Default argument for *readline()*

readline() now defaults to *ARGV if no argument is provided. (Rafael Garcia-Suarez)

state() **variables**

A new class of variables has been introduced. State variables are similar to my variables, but are declared with the state keyword in place of my. They're visible only in their lexical scope, but their value is persistent: unlike my variables, they're not undefined at scope entry, but retain their previous value. (Rafael

Garcia-Suarez, Nicholas Clark)

To use state variables, one needs to enable them by using

```
use feature 'state';
```

or by using the `-E` command-line switch in one-liners. See ‘‘Persistent Private Variables’’ in perlsub.

Stacked filetest operators

As a new form of syntactic sugar, it’s now possible to stack up filetest operators. You can now write `-f -w -x $file`in a ro w to mean `-x $file && -w _ && -f _`. See ‘‘-X’’ in perlfunc.

UNIVERSAL::DOES()

The `UNIVERSAL` class has a new method, `DOES()`. It has been added to solve semantic problems with the `isa()` method. `isa()` checks for inheritance, while `DOES()` has been designed to be overridden when module authors use other types of relations between classes (in addition to inheritance). (chromatic)

See ‘‘$obj->DOES(ROLE)’’ in UNIVERSAL.

Formats

Formats were improved in several ways. A new field, `^*`, can be used for variable-width, one-line-at-a-time text. Null characters are now handled correctly in picture lines. Using @# and `~~` together will now produce a compile-time error, as those format fields are incompatible. perlform(1) has been improved, and miscellaneous bugs fixed.

Byte-order modifiers for *pack()* and *unpack()*

There are two new byte-order modifiers, `>` (big-endian) and `<` (little-endian), that can be appended to most *pack()* and *unpack()* template characters and groups to force a certain byte-order for that type or group. See ‘‘pack’’ in perlfunc(1) and perlpacktut(1) for details.

`no VERSION`

You can now use `no` followed by a version number to specify that you want to use a version of perl older than the specified one.

`chdir`, `chmod` **and** `chown` **on filehandles**

`chdir`, `chmod` and `chown` can now work on filehandles as well as filenames, if the system supports respectively `fchdir`, `fchmod` and `fchown`, thanks to a patch provided by Gisle Aas.

OS groups

`$(` and `$)` now return groups in the order where the OS returns them, thanks to Gisle Aas. This wasn’t previously the case.

Recursive sort subs

You can now use recursive subroutines with *sort()*, thanks to Robin Houston.

Exceptions in constant folding

The constant folding routine is now wrapped in an exception handler, and if folding throws an exception (such as attempting to evaluate 0/0), perl now retains the current optree, rather than aborting the whole program. Without this change, programs would not compile if they had expressions that happened to generate exceptions, even though those expressions were in code that could never be reached at runtime. (Nicholas Clark, Dave Mitchell)

Source filters in `@INC`

It’s possible to enhance the mechanism of subroutine hooks in `@INC` by adding a source filter on top of the filehandle opened and returned by the hook. This feature was planned a long time ago, but wasn’t quite working until now. See ‘‘require’’ in perlfunc(1) for details. (Nicholas Clark)

New internal variables

`${^RE_DEBUG_FLAGS}`

This variable controls what debug flags are in effect for the regular expression engine when running under `use re "debug"`. See re for details.

`${^CHILD_ERROR_NATIVE}`
: This variable gives the native status returned by the last pipe close, backtick command, successful call to *wait()* or *waitpid()*, or from the *system()* operator. See perlvar(1) for details. (Contributed by Gisle Aas.)

`${^RE_TRIE_MAXBUF}`
: See ''Trie optimisation of literal string alternations''.

`${^WIN32_SLOPPY_STAT}`
: See ''Sloppy stat on Windows''.

Miscellaneous

`unpack()` now defaults to unpacking the `$_` variable.

`mkdir()` without arguments now defaults to `$_`.

The internal dump output has been improved, so that non-printable characters such as newline and backspace are output in `\x` notation, rather than octal.

The **-C** option can no longer be used on the `#!` line. It wasn't working there anyway, since the standard streams are already set up at this point in the execution of the perl interpreter. You can use *binmode()* instead to get the desired behaviour.

UCD 5.0.0

The copy of the Unicode Character Database included in Perl 5 has been updated to version 5.0.0.

MAD

MAD, which stands for *Miscellaneous Attribute Decoration*, is a still-in-development work leading to a Perl 5 to Perl 6 converter. To enable it, it's necessary to pass the argument `-Dmad` to Configure. The obtained perl isn't binary compatible with a regular perl 5.10, and has space and speed penalties; moreover not all regression tests still pass with it. (Larry Wall, Nicholas Clark)

kill() **on Windows**

On Windows platforms, `kill(-9, $pid)` now kills a process tree. (On Unix, this delivers the signal to all processes in the same process group.)

Incompatible Changes

Packing and UTF-8 strings

The semantics of *pack()* and *unpack()* regarding UTF-8-encoded data has been changed. Processing is now by default character per character instead of byte per byte on the underlying encoding. Notably, code that used things like `pack("a*", $string)` to see through the encoding of string will now simply get back the original `$string`. Packed strings can also get upgraded during processing when you store upgraded characters. You can get the old behaviour by using `use bytes`.

To be consistent with *pack()*, the `C0` in *unpack()* templates indicates that the data is to be processed in character mode, i.e. character by character; on the contrary, `U0` in *unpack()* indicates UTF-8 mode, where the packed string is processed in its UTF-8-encoded Unicode form on a byte by byte basis. This is reversed with regard to perl 5.8.X, but now consistent between *pack()* and *unpack()*.

Moreover, `C0` and `U0` can also be used in *pack()* templates to specify respectively character and byte modes.

`C0` and `U0` in the middle of a pack or unpack format now switch to the specified encoding mode, honoring parens grouping. Previously, parens were ignored.

Also, there is a new *pack()* character format, `W`, which is intended to replace the old `C`. `C` is kept for unsigned chars coded as bytes in the strings internal representation. `W` represents unsigned (logical) character values, which can be greater than 255. It is therefore more robust when dealing with potentially UTF-8-encoded data (as `C` will wrap values outside the range 0..255, and not respect the string encoding).

In practice, that means that pack formats are now encoding-neutral, except `C`.

For consistency, `A` in *unpack()* format now trims all Unicode whitespace from the end of the string. Before perl 5.9.2, it used to strip only the classical ASCII space characters.

Byte/character count feature in *unpack()*

A new *unpack()* template character, `"."`, returns the number of bytes or characters (depending on the selected encoding mode, see above) read so far.

The `$*` and `$#` variables have been removed

`$*`, which was deprecated in favor of the `/s` and `/m` regexp modifiers, has been removed.

The deprecated `$#` variable (output format for numbers) has been removed.

Two new severe warnings, `$#/$* is no longer supported`, have been added.

substr() **lvalues are no longer fixed-length**

The lvalues returned by the three argument form of *substr()* used to be a ''fixed length window'' on the original string. In some cases this could cause surprising action at distance or other undefined behaviour. Now the length of the window adjusts itself to the length of the string assigned to it.

Parsing of `-f _`

The identifier _ is now forced to be a bareword after a filetest operator. This solves a number of misparsing issues when a global _ subroutine is defined.

`:unique`

The `:unique` attribute has been made a no-op, since its current implementation was fundamentally flawed and not threadsafe.

Effect of pragmas in eval

The compile-time value of the `%^H` hint variable can now propagate into eval("")uated code. This makes it more useful to implement lexical pragmas.

As a side-effect of this, the overloaded-ness of constants now propagates into eval("").

chdir FOO

A bareword argument to *chdir()* is now recognized as a file handle. Earlier releases interpreted the bareword as a directory name. (Gisle Aas)

Handling of .pmc files

An old feature of perl was that before `require` or `use` look for a file with a *.pm* extension, they will first look for a similar filename with a *.pmc* extension. If this file is found, it will be loaded in place of any potentially existing file ending in a *.pm* extension.

Previously, *.pmc* files were loaded only if more recent than the matching *.pm* file. Starting with 5.9.4, they'll be always loaded if they exist.

$^V is now a `version` object instead of a v-string

$^V can still be used with the `%vd` format in printf, but any character-level operations will now access the string representation of the `version` object and not the ordinals of a v-string. Expressions like `substr($^V, 0, 2)`or `split //, $^V` no longer w ork and must be rewritten.

@− and @+ in patterns

The special arrays `@-` and `@+` are no longer interpolated in regular expressions. (Sadahiro Tomoyuki)

`$AUTOLOAD` **can now be tainted**

If you call a subroutine by a tainted name, and if it defers to an AUTOLOAD function, then `$AUTOLOAD` will be (correctly) tainted. (Rick Delaney)

Tainting and printf

When perl is run under taint mode, `printf()` and `sprintf()` will now reject any tainted format argument. (Rafael Garcia-Suarez)

undef and signal handlers

Undefining or deleting a signal handler via `undef $SIG{FOO}` is now equivalent to setting it to `'DEFAULT'`. (Rafael Garcia-Suarez)

strictures and dereferencing in *defined()*

`use strict 'refs'`w as ignoring taking a hard reference in an argument to *defined()*, as in :

```
use strict 'refs';
my $x = 'foo';
if (defined $$x) {...}
```

This now correctly produces the run-time error `Can't use string as a SCALAR ref while "strict refs" in use`.

`defined @$foo`and `defined %$bar` are no w also subject to `strict 'refs'` (that is, `$foo` and `$bar` shall be proper references there.) (`defined(@foo)` and `defined(%bar)` are discouraged constructs anyway.) (Nicholas Clark)

`(?p{})` has been removed

The regular expression construct `(?p{})`, which was deprecated in perl 5.8, has been removed. Use `(??{})` instead. (Rafael Garcia-Suarez)

Pseudo-hashes have been removed

Support for pseudo-hashes has been removed from Perl 5.9. (The `fields` pragma remains here, but uses an alternate implementation.)

Removal of the bytecode compiler and of perlcc

`perlcc` the byteloader and the supporting modules (B::C, B::CC, B::Bytecode, etc.) are no longer distributed with the perl sources. Those experimental tools have never worked reliably, and, due to the lack of volunteers to keep them in line with the perl interpreter developments, it was decided to remove them instead of shipping a broken version of those. The last version of those modules can be found with perl 5.9.4.

However the B compiler framework stays supported in the perl core, as with the more useful modules it has permitted (among others, B::Deparse and B::Concise).

Removal of the JPL

The JPL (Java-Perl Lingo) has been removed from the perl sources tarball.

Recursive inheritance detected earlier

Perl will now immediately throw an exception if you modify any package's `@ISA` in such a way that it would cause recursive inheritance.

Previously, the exception would not occur until Perl attempted to make use of the recursive inheritance while resolving a method or doing a `$foo->isa($bar)` lookup.

warnings::enabled and warnings::warnif changed to favor users of modules

The behaviour in 5.10.x favors the person using the module; The behaviour in 5.8.x favors the module writer;

Assume the following code:

```
main calls Foo::Bar::baz()
Foo::Bar inherits from Foo::Base
Foo::Bar::baz() calls Foo::Base::_bazbaz()
Foo::Base::_bazbaz() calls: warnings::warnif('substr', 'some warning
message');
```

On 5.8.x, the code warns when Foo::Bar contains `use warnings;` It does not matter if Foo::Base or main have warnings enabled to disable the warning one has to modify Foo::Bar.

On 5.10.0 and newer, the code warns when main contains `use warnings;` It does not matter if Foo::Base or Foo::Bar have warnings enabled to disable the warning one has to modify main.

Modules and Pragmata

Upgrading individual core modules

Even more core modules are now also available separately through the CPAN. If you wish to update one of these modules, you don't need to wait for a new perl release. From within the cpan shell, running the 'r' command will report on modules with upgrades available. See `perldoc(1)` `CPAN`for more information.

Pragmata Changes

`feature`
: The new pragma `feature` is used to enable new features that might break old code. See "The `feature` pragma" above.

`mro`
: This new pragma enables to change the algorithm used to resolve inherited methods. See "New Pragma, `mro`" above.

Scoping of the `sort` pragma
: The `sort` pragma is now lexically scoped. Its effect used to be global.

Scoping of `bignum`, `bigint`, `bigrat`
: The three numeric pragmas `bignum`, `bigint` and `bigrat` are now lexically scoped. (Tels)

`base`
: The `base` pragma now warns if a class tries to inherit from itself. (Curtis ‘‘Ovid’’ Poe)

`strict` and `warnings`
: `strict` and `warnings` will now complain loudly if they are loaded via incorrect casing (as in `use Strict;`). (Johan Vromans)

`version`
: The `version` module provides support for version objects.

`warnings`
: The `warnings` pragma doesn’t load `Carp` anymore. That means that code that used `Carp` routines without having loaded it at compile time might need to be adjusted; typically, the following (faulty) code won’t work anymore, and will require parentheses to be added after the function name:

```
use warnings;
require Carp;
Carp::confess 'argh';
```

`less`
: `less` now does something useful (or at least it tries to). In fact, it has been turned into a lexical pragma. So, in your modules, you can now test whether your users have requested to use less CPU, or less memory, less magic, or maybe even less fat. See less for more. (Joshua ben Jore)

New modules

- `encoding::warnings` by Audrey Tang, is a module to emit warnings whenever an ASCII character string containing high-bit bytes is implicitly converted into UTF-8. It’s a lexical pragma since Perl 5.9.4; on older perls, its effect is global.
- `Module::CoreList` by Richard Clamp, is a small handy module that tells you what versions of core modules ship with any versions of Perl 5. It comes with a command-line frontend, `corelist`.
- `Math::BigInt::FastCalc` is an XS-enabled, and thus faster, version of `Math::BigInt::Calc`
- `Compress::Zlib` is an interface to the zlib compression library. It comes with a bundled version of zlib, so having a working zlib is not a prerequisite to install it. It’s used by `Archive::Tar` (see below).
- `IO::Zlib` is an `IO::`-style interface to `Compress::Zlib`
- `Archive::Tar` is a module to manipulate `tar` archives.
- `Digest::SHA` is a module used to calculate many types of SHA digests, has been included for SHA support in the CPAN module.
- `ExtUtils::CBuilder` and `ExtUtils::ParseXS` have been added.
- `Hash::Util::FieldHash` by Anno Siegel, has been added. This module provides support for *field hashes*: hashes that maintain an association of a reference with a value, in a thread-safe garbage-

collected way. Such hashes are useful to implement inside-out objects.

- `Module::Build` by Ken Williams, has been added. It's an alternative to `ExtUtils::MakeMaker` to build and install perl modules.
- `Module::Load` by Jos Boumans, has been added. It provides a single interface to load Perl modules and *.pl* files.
- `Module::Loaded` by Jos Boumans, has been added. It's used to mark modules as loaded or unloaded.
- `Package::Constants` by Jos Boumans, has been added. It's a simple helper to list all constants declared in a given package.
- `Win32API::File` by Tye McQueen, has been added (for Windows builds). This module provides low-level access to Win32 system API calls for files/dirs.
- `Locale::Maketext::Simple` needed by CPANPLUS, is a simple wrapper around `Locale::Maketext::Lexicon` Note that `Locale::Maketext::Lexicon` isn't included in the perl core; the behaviour of `Locale::Maketext::Simple` gracefully degrades when the later isn't present.
- `Params::Check` implements a generic input parsing/checking mechanism. It is used by CPANPLUS.
- `Term::UI` simplifies the task to ask questions at a terminal prompt.
- `Object::Accessor` provides an interface to create per-object accessors.
- `Module::Pluggable` is a simple framework to create modules that accept pluggable sub-modules.
- `Module::Load::Conditional` provides simple ways to query and possibly load installed modules.
- `Time::Piece` provides an object oriented interface to time functions, overriding the built-ins *localtime()* and *gmtime()*.
- `IPC::Cmd` helps to find and run external commands, possibly interactively.
- `File::Fetch` provide a simple generic file fetching mechanism.
- `Log::Message` and `Log::Message::Simple` are used by the log facility of `CPANPLUS`.
- `Archive::Extract` is a generic archive extraction mechanism for *.tar* (plain, gzipped or bzipped) or *.zip* files.
- `CPANPLUS` provides an API and a command-line tool to access the CPAN mirrors.
- `Pod::Escapes` provides utilities that are useful in decoding Pod E<...> sequences.
- `Pod::Simple` is now the backend for several of the Pod-related modules included with Perl.

Selected Changes to Core Modules

`Attribute::Handlers`

`Attribute::Handlers` can now report the caller's file and line number. (David Feldman)

All interpreted attributes are now passed as array references. (Damian Conway)

`B::Lint`

`B::Lint` is now based on `Module::Pluggable` and so can be extended with plugins. (Joshua ben Jore)

`B` It's now possible to access the lexical pragma hints (`%^H`) by using the method *B::COP::hints_hash()*. It returns a `B::RHE` object, which in turn can be used to get a hash reference via the method *B::RHE::HASH()*. (Joshua ben Jore)

`Thread`

As the old 5005thread threading model has been removed, in favor of the ithreads scheme, the `Thread` module is now a compatibility wrapper, to be used in old code only. It has been removed from the default list of dynamic extensions.

Utility Changes

perl -d

The Perl debugger can now save all debugger commands for sourcing later; notably, it can now emulate stepping backwards, by restarting and rerunning all bar the last command from a saved command history.

It can also display the parent inheritance tree of a given class, with the `i` command.

ptar

`ptar` is a pure perl implementation of `tar` that comes with `Archive::Tar`

ptardiff

`ptardiff` is a small utility used to generate a diff between the contents of a tar archive and a directory tree. Like `ptar`, it comes with `Archive::Tar`

shasum

`shasum` is a command-line utility, used to print or to check SHA digests. It comes with the new `Digest::SHA` module.

corelist

The `corelist` utility is now installed with perl (see ''New modules'' above).

h2ph and h2xs

`h2ph` and `h2xs` have been made more robust with regard to ''modern'' C code.

`h2xs` implements a new option `--use-xsloader` to force use of `XSLoader` even in backwards compatible modules.

The handling of authors' names that had apostrophes has been fixed.

Any enums with negative values are now skipped.

perlivp

`perlivp(1)` no longer checks for *.*ph* files by default. Use the new `-a` option to run *all* tests.

find2perl

`find2perl` now assumes `-print` as a default action. Previously, it needed to be specified explicitly.

Several bugs have been fixed in `find2perl`, regarding `-exec` and `-eval`. Also the options `-path`, `-ipath` and `-iname` have been added.

config_data

`config_data` is a new utility that comes with `Module::Build` It provides a command-line interface to the configuration of Perl modules that use Module::Build's framework of configurability (that is, `*::ConfigData` modules that contain local configuration information for their parent modules.)

cpanp

`cpanp`, the CPANPLUS shell, has been added. (`cpanp-run-perl`, a helper for CPANPLUS operation, has been added too, but isn't intended for direct use).

cpan2dist

`cpan2dist` is a new utility that comes with CPANPLUS. It's a tool to create distributions (or packages) from CPAN modules.

pod2html

The output of `pod2html` has been enhanced to be more customizable via CSS. Some formatting problems were also corrected. (Jari Aalto)

New Documentation

The perlpragma(1) manpage documents how to write one's own lexical pragmas in pure Perl (something that is possible starting with 5.9.4).

The new perlglossary(1) manpage is a glossary of terms used in the Perl documentation, technical and

otherwise, kindly provided by O'Reilly Media, Inc.

The perlreguts(1) manpage, courtesy of Yves Orton, describes internals of the Perl regular expression engine.

The perlreapi(1) manpage describes the interface to the perl interpreter used to write pluggable regular expression engines (by Ævar Arnfjörð Bjarmason).

The perlunitut(1) manpage is an tutorial for programming with Unicode and string encodings in Perl, courtesy of Juerd Waalboer.

A new manual page, perlunifaq(1) (the Perl Unicode FAQ), has been added (Juerd Waalboer).

The perlcommunity(1) manpage gives a description of the Perl community on the Internet and in real life. (Edgar ''Trizor'' Bering)

The CORE manual page documents the `CORE::` namespace. (Tels)

The long-existing feature of `/(?{...})/` regexps setting `$_` and *pos()* is now documented.

Performance Enhancements

In-place sorting

Sorting arrays in place (`@a = sort @a`) is now optimized to avoid making a temporary copy of the array.

Likewise, `reverse sort ...` is now optimized to sort in reverse, avoiding the generation of a temporary intermediate list.

Lexical array access

Access to elements of lexical arrays via a numeric constant between 0 and 255 is now faster. (This used to be only the case for global arrays.)

XS-assisted SWASHGET

Some pure-perl code that perl was using to retrieve Unicode properties and transliteration mappings has been reimplemented in XS.

Constant subroutines

The interpreter internals now support a far more memory efficient form of inlineable constants. Storing a reference to a constant value in a symbol table is equivalent to a full typeglob referencing a constant subroutine, but using about 400 bytes less memory. This proxy constant subroutine is automatically upgraded to a real typeglob with subroutine if necessary. The approach taken is analogous to the existing space optimisation for subroutine stub declarations, which are stored as plain scalars in place of the full typeglob.

Several of the core modules have been converted to use this feature for their system dependent constants - as a result `use POSIX;` now takes about 200K less memory.

PERL_DONT_CREATE_GVSV

The new compilation flag `PERL_DONT_CREATE_GVSV`, introduced as an option in perl 5.8.8, is turned on by default in perl 5.9.3. It prevents perl from creating an empty scalar with every new typeglob. See perl589delta(1) for details.

Weak references are cheaper

Weak reference creation is now *O(1)* rather than *O(n)*, courtesy of Nicholas Clark. Weak reference deletion remains *O(n)*, but if deletion only happens at program exit, it may be skipped completely.

sort() enhancements

Salvador Fandiño provided improvements to reduce the memory usage of `sort` and to speed up some cases.

Memory optimisations

Several internal data structures (typeglobs, GVs, CVs, formats) have been restructured to use less memory. (Nicholas Clark)

UTF-8 cache optimisation

The UTF-8 caching code is now more efficient, and used more often. (Nicholas Clark)

Sloppy stat on Windows

On Windows, perl's *stat()* function normally opens the file to determine the link count and update attributes that may have been changed through hard links. Setting ${^WIN32_SLOPPY_STAT} to a true value speeds up *stat()* by not performing this operation. (Jan Dubois)

Regular expressions optimisations

Engine de-recursivised

The regular expression engine is no longer recursive, meaning that patterns that used to overflow the stack will either die with useful explanations, or run to completion, which, since they were able to blow the stack before, will likely take a very long time to happen. If you were experiencing the occasional stack overflow (or segfault) and upgrade to discover that now perl apparently hangs instead, look for a degenerate regex. (Dave Mitchell)

Single char char-classes treated as literals

Classes of a single character are now treated the same as if the character had been used as a literal, meaning that code that uses char-classes as an escaping mechanism will see a speedup. (Yves Orton)

Trie optimisation of literal string alternations

Alternations, where possible, are optimised into more efficient matching structures. String literal alternations are merged into a trie and are matched simultaneously. This means that instead of O(N) time for matching N alternations at a given point, the new code performs in O(1) time. A new special variable, ${^RE_TRIE_MAXBUF}, has been added to fine-tune this optimization. (Yves Orton)

Note: Much code exists that works around perl's historic poor performance on alternations. Often the tricks used to do so will disable the new optimisations. Hopefully the utility modules used for this purpose will be educated about these new optimisations.

Aho-Corasick start-point optimisation

When a pattern starts with a trie-able alternation and there aren't better optimisations available, the regex engine will use Aho-Corasick matching to find the start point. (Yves Orton)

Installation and Configuration Improvements

Configuration improvements

`-Dusesitecustomize`

Run-time customization of `@INC` can be enabled by passing the `-Dusesitecustomize` flag to Configure. When enabled, this will make perl run *`$sitelibexp`/sitecustomize.pl* before anything else. This script can then be set up to add additional entries to `@INC`.

Relocatable installations

There is now Configure support for creating a relocatable perl tree. If you Configure with `-Duserelocatableinc`, then the paths in `@INC` (and everything else in `%Config`) can be optionally located via the path of the perl executable.

That means that, if the string `"../"` is found at the start of any path, it's substituted with the directory of $^X. So, the relocation can be configured on a per-directory basis, although the default with `-Duserelocatableinc` is that everything is relocated. The initial install is done to the original configured prefix.

strlcat() and *strlcpy()*

The configuration process now detects whether *strlcat()* and *strlcpy()* are available. When they are not available, perl's own version is used (from Russ Allbery's public domain implementation). Various places in the perl interpreter now use them. (Steve Peters)

`d_pseudofork` and `d_printf_format_null`

A new configuration variable, available as `$Config{d_pseudofork}` in the Config module, has been added, to distinguish real *fork()* support from fake pseudofork used on Windows platforms.

A new configuration variable, `d_printf_format_null`, has been added, to see if printf-like

formats are allowed to be NULL.

Configure help
`Configure -h` has been extended with the most commonly used options.

Compilation improvements

Parallel build
Parallel makes should work properly now, although there may still be problems if `make test` is instructed to run in parallel.

Borland's compilers support
Building with Borland's compilers on Win32 should work more smoothly. In particular Steve Hay has worked to side step many warnings emitted by their compilers and at least one C compiler internal error.

Static build on Windows
Perl extensions on Windows now can be statically built into the Perl DLL.

Also, it's now possible to build a `perl-static.exe` that doesn't depend on the Perl DLL on Win32. See the Win32 makefiles for details. (Vadim Konovalov)

ppport.h files
All *ppport.h* files in the XS modules bundled with perl are now autogenerated at build time. (Marcus Holland-Moritz)

C++ compatibility
Efforts have been made to make perl and the core XS modules compilable with various C++ compilers (although the situation is not perfect with some of the compilers on some of the platforms tested.)

Support for Microsoft 64-bit compiler
Support for building perl with Microsoft's 64-bit compiler has been improved. (ActiveState)

Visual C++
Perl can now be compiled with Microsoft Visual C++ 2005 (and 2008 Beta 2).

Win32 builds
All win32 builds (MS-Win, WinCE) have been merged and cleaned up.

Installation improvements

Module auxiliary files
README files and changelogs for CPAN modules bundled with perl are no longer installed.

New Or Improved Platforms

Perl has been reported to work on Symbian OS. See perlsymbian(1) for more information.

Many improvements have been made towards making Perl work correctly on z/OS.

Perl has been reported to work on DragonFlyBSD and MidnightBSD.

Perl has also been reported to work on NexentaOS (http://www.gnusolaris.org/).

The VMS port has been improved. See perlvms.

Support for Cray XT4 Catamount/Qk has been added. See *hints/catamount.sh* in the source code distribution for more information.

Vendor patches have been merged for RedHat and Gentoo.

DynaLoader::dl_unload_file() now works on Windows.

Selected Bug Fixes

strictures in regexp-eval blocks
`strict` wasn't in effect in regexp-eval blocks (`/(?{...})/`).

Calling *CORE::require()*
CORE::require() and *CORE::do()* were always parsed as *require()* and *do()* when they were overridden. This is now fixed.

Subscripts of slices
: You can now use a non-arrowed form for chained subscripts after a list slice, like in:

```
({foo => "bar"})[0]{foo}
```

: This used to be a syntax error; a `->` was required.

`no warnings 'category'` works correctly with -w
: Previously when running with warnings enabled globally via `-w`, selective disabling of specific warning categories would actually turn off all warnings. This is now fixed; now `no warnings 'io';` will only turn off warnings in the `io` class. Previously it would erroneously turn off all warnings.

threads improvements
: Several memory leaks in ithreads were closed. Also, ithreads were made less memory-intensive.

: `threads` is now a dual-life module, also available on CPAN. It has been expanded in many ways. A *kill()* method is available for thread signalling. One can get thread status, or the list of running or joinable threads.

: A new `threads->exit()` method is used to exit from the application (this is the default for the main thread) or from the current thread only (this is the default for all other threads). On the other hand, the *exit()* built-in now always causes the whole application to terminate. (Jerry D. Hedden)

chr() and negative values
: *chr()* on a negative value now gives `\x{FFFD}`, the Unicode replacement character, unless when the `bytes` pragma is in effect, where the low eight bits of the value are used.

PERL5SHELL and tainting
: On Windows, the PERL5SHELL environment variable is now checked for taintedness. (Rafael Garcia-Suarez)

Using *FILE{IO}
: `stat()` and `-X` filetests now treat *FILE{IO} filehandles like *FILE filehandles. (Steve Peters)

Overloading and reblessing
: Overloading now works when references are reblessed into another class. Internally, this has been implemented by moving the flag for ‘‘overloading’’ from the reference to the referent, which logically is where it should always have been. (Nicholas Clark)

Overloading and UTF-8
: A few bugs related to UTF-8 handling with objects that have stringification overloaded have been fixed. (Nicholas Clark)

eval memory leaks fixed
: Traditionally, `eval 'syntax error'` has leaked badly. Many (but not all) of these leaks have now been eliminated or reduced. (Dave Mitchell)

Random device on Windows
: In previous versions, perl would read the file */dev/urandom* if it existed when seeding its random number generator. That file is unlikely to exist on Windows, and if it did would probably not contain appropriate data, so perl no longer tries to read it on Windows. (Alex Davies)

PERLIO_DEBUG
: The `PERLIO_DEBUG` environment variable no longer has any effect for setuid scripts and for scripts run with **-T**.

: Moreover, with a thread-enabled perl, using `PERLIO_DEBUG` could lead to an internal buffer overflow. This has been fixed.

PerlIO::scalar and read-only scalars
: PerlIO::scalar will now prevent writing to read-only scalars. Moreover, *seek()* is now supported with PerlIO::scalar-based filehandles, the underlying string being zero-filled as needed. (Rafael, Jarkko Hietaniemi)

study() and UTF-8
: *study()* never worked for UTF-8 strings, but could lead to false results. It's now a no-op on UTF-8 data. (Yves Orton)

Critical signals
: The signals SIGILL, SIGBUS and SIGSEGV are now always delivered in an ''unsafe'' manner (contrary to other signals, that are deferred until the perl interpreter reaches a reasonably stable state; see ''Deferred Signals (Safe Signals)'' in perlipc). (Rafael)

`@INC`-hook fix
: When a module or a file is loaded through an `@INC`-hook, and when this hook has set a filename entry in `%INC`, `__FILE__` is now set for this module accordingly to the contents of that `%INC` entry. (Rafael)

`-t` switch fix
: The `-w` and `-t` switches can now be used together without messing up which categories of warnings are activated. (Rafael)

Duping UTF-8 filehandles
: Duping a filehandle which has the `:utf8` PerlIO layer set will now properly carry that layer on the duped filehandle. (Rafael)

Localisation of hash elements
: Localizing a hash element whose key was given as a variable didn't work correctly if the variable was changed while the *local()* was in effect (as in `local $h{$x}; ++$x`). (Bo Lindbergh)

New or Changed Diagnostics

Use of uninitialized value
: Perl will now try to tell you the name of the variable (if any) that was undefined.

Deprecated use of *my()* in false conditional
: A new deprecation warning, *Deprecated use of my() in false conditional*, has been added, to warn against the use of the dubious and deprecated construct

```
my $x if 0;
```

: See perldiag. Use `state` variables instead.

!=~ should be !~
: A new warning, `!=~ should be !~`, is emitted to prevent this misspelling of the non-matching operator.

Newline in left-justified string
: The warning *Newline in left-justified string* has been removed.

Too late for ''-T'' option
: The error *Too late for "-T" option* has been reformulated to be more descriptive.

''%s'' variable `%s` masks earlier declaration
: This warning is now emitted in more consistent cases; in short, when one of the declarations involved is a `my` variable:

```
my $x; my $x; # warns
my $x; our $x; # warns
our $x; my $x; # warns
```

: On the other hand, the following:

```
our $x; our $x;
```

: now gives a `"our" variable %s redeclared` warning.

readdir()/*closedir()*/etc. attempted on invalid dirhandle
: These new warnings are now emitted when a dirhandle is used but is either closed or not really a dirhandle.

Opening dirhandle/filehandle %s also as a file/directory
: Two deprecation warnings have been added: (Rafael)

```
Opening dirhandle %s also as a file
Opening filehandle %s also as a directory
```

Use of -P is deprecated
: Perl's command-line switch `-P` is now deprecated.

v-string in use/require is non-portable
: Perl will warn you against potential backwards compatibility problems with the `use VERSION` syntax.

perl -V
: `perl -V`has se veral improvements, making it more useable from shell scripts to get the value of configuration variables. See perlrun(1) for details.

Changed Internals

In general, the source code of perl has been refactored, tidied up, and optimized in many places. Also, memory management and allocation has been improved in several points.

When compiling the perl core with gcc, as many gcc warning flags are turned on as is possible on the platform. (This quest for cleanliness doesn't extend to XS code because we cannot guarantee the tidiness of code we didn't write.) Similar strictness flags have been added or tightened for various other C compilers.

Reordering of SVt_* constants

The relative ordering of constants that define the various types of `SV` have changed; in particular, `SVt_PVGV` has been moved before `SVt_PVLV`, `SVt_PVAV`, `SVt_PVHV` and `SVt_PVCV`. This is unlikely to make any difference unless you have code that explicitly makes assumptions about that ordering. (The inheritance hierarchy of `B::*` objects has been changed to reflect this.)

Elimination of SVt_PVBM

Related to this, the internal type `SVt_PVBM` has been removed. This dedicated type of `SV` was used by the `index` operator and parts of the regexp engine to facilitate fast Boyer-Moore matches. Its use internally has been replaced by `SV`s of type `SVt_PVGV`.

New type SVt_BIND

A new type `SVt_BIND` has been added, in readiness for the project to implement Perl 6 on 5. There deliberately is no implementation yet, and they cannot yet be created or destroyed.

Removal of CPP symbols

The C preprocessor symbols `PERL_PM_APIVERSION` and `PERL_XS_APIVERSION`, which were supposed to give the version number of the oldest perl binary-compatible (resp. source-compatible) with the present one, were not used, and sometimes had misleading values. They have been removed.

Less space is used by ops

The `BASEOP` structure now uses less space. The `op_seq` field has been removed and replaced by a single bit bit-field `op_opt`. `op_type` is now 9 bits long. (Consequently, the `B::OP` class doesn't provide an `seq` method anymore.)

New parser

perl's parser is now generated by bison (it used to be generated by byacc.) As a result, it seems to be a bit more robust.

Also, Dave Mitchell improved the lexer debugging output under `-DT`.

Use of `const`

Andy Lester supplied many improvements to determine which function parameters and local variables could actually be declared `const` to the C compiler. Steve Peters provided new `*_set` macros and reworked the core to use these rather than assigning to macros in LVALUE context.

Mathoms

A new file, *mathoms.c*, has been added. It contains functions that are no longer used in the perl core, but that remain available for binary or source compatibility reasons. However, those functions will not be compiled in if you add `-DNO_MATHOMS` in the compiler flags.

`AvFLAGS` **has been removed**

The `AvFLAGS` macro has been removed.

`av_*` **changes**

The `av_*()` functions, used to manipulate arrays, no longer accept null `AV*` parameters.

$^H and %^H

The implementation of the special variables $^H and %^H has changed, to allow implementing lexical pragmas in pure Perl.

B:: modules inheritance changed

The inheritance hierarchy of `B::` modules has changed; `B::NV` now inherits from `B::SV` (it used to inherit from `B::IV`

Anonymous hash and array constructors

The anonymous hash and array constructors now take 1 op in the optree instead of 3, now that pp_anonhash and pp_anonlist return a reference to an hash/array when the op is flagged with OPf_SPECIAL. (Nicholas Clark)

Known Problems

There's still a remaining problem in the implementation of the lexical `$_`: it doesn't work inside `/(?{...})/` blocks. (See the TODO test in *t/op/mydef.t*.)

Stacked filetest operators won't work when the `filetest` pragma is in effect, because they rely on the *stat()* buffer _ being populated, and filetest bypasses *stat()*.

UTF-8 problems

The handling of Unicode still is unclean in several places, where it's dependent on whether a string is internally flagged as UTF-8. This will be made more consistent in perl 5.12, but that won't be possible without a certain amount of backwards incompatibility.

Platform Specific Problems

When compiled with g++ and thread support on Linux, it's reported that the `$!` stops working correctly. This is related to the fact that the glibc provides two *strerror_r(3)* implementation, and perl selects the wrong one.

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/rt3/ 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.

SEE ALSO

The *Changes* file and the perl590delta to perl595delta man pages for 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.