NAME

perl5005delta - what's new for perl5.005

DESCRIPTION

This document describes differences between the 5.004 release and this one.

About the new versioning system

Perl is now developed on two tracks: a maintenance track that makes small, safe updates to released production versions with emphasis on compatibility; and a development track that pursues more aggressive evolution. Maintenance releases (which should be considered production quality) have subversion numbers that run from `1` to `49`, and development releases (which should be considered ''alpha'' quality) run from `50` to `99`.

Perl 5.005 is the combined product of the new dual-track development scheme.

Incompatible Changes

WARNING: This version is not binary compatible with Perl 5.004.

Starting with Perl 5.004_50 there were many deep and far-reaching changes to the language internals. If you have dynamically loaded extensions that you built under perl 5.003 or 5.004, you can continue to use them with 5.004, but you will need to rebuild and reinstall those extensions to use them 5.005. See *INSTALL* for detailed instructions on how to upgrade.

Default installation structure has changed

The new Configure defaults are designed to allow a smooth upgrade from 5.004 to 5.005, but you should read *INSTALL* for a detailed discussion of the changes in order to adapt them to your system.

Perl Source Compatibility

When none of the experimental features are enabled, there should be very few user-visible Perl source compatibility issues.

If threads are enabled, then some caveats apply. `@_` and `$_` become lexical variables. The effect of this should be largely transparent to the user, but there are some boundary conditions under which user will need to be aware of the issues. For example, `local(@_)` results in a ''Can't localize lexical variable `@_` ...'' message. This may be enabled in a future version.

Some new keywords have been introduced. These are generally expected to have very little impact on compatibility. See "New `INIT` keyword", "New `lock` keyword", and "New `qr//` operator".

Certain barewords are now reserved. Use of these will provoke a warning if you have asked for them with the `-w` switch. See "`our` is now a reserved word".

C Source Compatibility

There have been a large number of changes in the internals to support the new features in this release.

- Core sources now require ANSI C compiler

  An ANSI C compiler is now **required** to build perl. See *INSTALL*.

- All Perl global variables must now be referenced with an explicit prefix

  All Perl global variables that are visible for use by extensions now have a `PL_` prefix. New extensions should `not` refer to perl globals by their unqualified names. To preserve sanity, we provide limited backward compatibility for globals that are being widely used like `sv_undef` and `na` (which should now be written as `PL_sv_undef`, `PL_na` etc.)

  If you find that your XS extension does not compile anymore because a perl global is not visible, try adding a `PL_` prefix to the global and rebuild.

  It is strongly recommended that all functions in the Perl API that don't begin with `perl` be referenced with a `Perl_` prefix. The bare function names without the `Perl_` prefix are supported with macros, but this support may cease in a future release.

  See perlapi.

- Enabling threads has source compatibility issues

  Perl built with threading enabled requires extensions to use the new `dTHR` macro to initialize the handle to access per-thread data. If you see a compiler error that talks about the variable `thr` not being declared (when building a module that has XS code), you need to add `dTHR;` at the beginning of the block that elicited the error.

  The API function `perl_get_sv("@",GV_ADD)` should be used instead of directly accessing perl globals as `GvSV(errgv)`. The API call is backward compatible with existing perls and provides source compatibility with threading is enabled.

  See ''C Source Compatibility'' for more information.

Binary Compatibility

This version is NOT binary compatible with older versions. All extensions will need to be recompiled. Further binaries built with threads enabled are incompatible with binaries built without. This should largely be transparent to the user, as all binary incompatible configurations have their own unique architecture name, and extension binaries get installed at unique locations. This allows coexistence of several configurations in the same directory hierarchy. See *INSTALL*.

Security fixes may affect compatibility

A few taint leaks and taint omissions have been corrected. This may lead to ''failure'' of scripts that used to work with older versions. Compiling with -DINCOMPLETE_TAINTS provides a perl with minimal amounts of changes to the tainting behavior. But note that the resulting perl will have known insecurities.

Oneliners with the `-e` switch do not create temporary files anymore.

Relaxed new mandatory warnings introduced in 5.004

Many new warnings that were introduced in 5.004 have been made optional. Some of these warnings are still present, but perl's new features make them less often a problem. See ''New Diagnostics''.

Licensing

Perl has a new Social Contract for contributors. See *Porting/Contract*.

The license included in much of the Perl documentation has changed. Most of the Perl documentation was previously under the implicit GNU General Public License or the Artistic License (at the user's choice). Now much of the documentation unambiguously states the terms under which it may be distributed. Those terms are in general much less restrictive than the GNU GPL. See perl and the individual perl manpages listed therein.

Core Changes

Threads

WARNING: Threading is considered an **experimental** feature. Details of the implementation may change without notice. There are known limitations and some bugs. These are expected to be fixed in future versions.

See *README.threads*.

Compiler

WARNING: The Compiler and related tools are considered **experimental**. Features may change without notice, and there are known limitations and bugs. Since the compiler is fully external to perl, the default configuration will build and install it.

The Compiler produces three different types of transformations of a perl program. The C backend generates C code that captures perl's state just before execution begins. It eliminates the compile-time overheads of the regular perl interpreter, but the run-time performance remains comparatively the same. The CC backend generates optimized C code equivalent to the code path at run-time. The CC backend has greater potential for big optimizations, but only a few optimizations are implemented currently. The Bytecode backend generates a platform independent bytecode representation of the interpreter's state just before execution. Thus, the Bytecode back end also eliminates much of the compilation overhead of the interpreter.

The compiler comes with several valuable utilities.

`B::Lint` is an experimental module to detect and warn about suspicious code, especially the cases that the `-w` switch does not detect.

`B::Deparse` can be used to demystify perl code, and understand how perl optimizes certain constructs.

`B::Xref` generates cross reference reports of all definition and use of variables, subroutines and formats in a program.

`B::Showlex` show the lexical variables used by a subroutine or file at a glance.

`perlcc` is a simple frontend for compiling perl.

See `ext/B/README`, B, and the respective compiler modules.

Regular Expressions

Perl's regular expression engine has been seriously overhauled, and many new constructs are supported. Several bugs have been fixed.

Here is an itemized summary:

Many new and improved optimizations

Changes in the RE engine:

```
Unneeded nodes removed;
Substrings merged together;
New types of nodes to process (SUBEXPR)* and similar expressions
quickly, used if the SUBEXPR has no side effects and matches
strings of the same length;
Better optimizations by lookup for constant substrings;
Better search for constants substrings anchored by $ ;
```

Changes in Perl code using RE engine:

```
More optimizations to s/longer/short/;
study() was not working;
/blah/ may be optimized to an analogue of index() if $& $` $' not seen;
Unneeded copying of matched-against string removed;
Only matched part of the string is copying if $` $' were not seen;
```

Many bug fixes

Note that only the major bug fixes are listed here. See *Changes* for others.

```
Backtracking might not restore start of $3.
No feedback if max count for * or + on "complex" subexpression
was reached, similarly (but at compile time) for {3,34567}
Primitive restrictions on max count introduced to decrease a
possibility of a segfault;
(ZERO-LENGTH)* could segfault;
(ZERO-LENGTH)* was prohibited;
Long REs were not allowed;
/RE/g could skip matches at the same position after a
zero-length match;
```

New regular expression constructs

The following new syntax elements are supported:

```
(?<=RE)
(?<!RE)
(?{ CODE })
(?i-x)
(?i:RE)
(?(COND)YES_RE|NO_RE)
(?>RE)
\z
```

New operator for precompiled regular expressions
See "New `qr//` operator".

Other improvements

```
Better debugging output (possibly with colors),
even from non-debugging Perl;
RE engine code now looks like C, not like assembler;
Behaviour of RE modifiable by `use re' directive;
Improved documentation;
Test suite significantly extended;
Syntax [:^upper:] etc., reserved inside character classes;
```

Incompatible changes

```
(?i) localized inside enclosing group;
$( is not interpolated into RE any more;
/RE/g may match at the same position (with non-zero length)
after a zero-length match (bug fix).
```

See perlre(1) and perlop.

Improved *malloc()*

See banner at the beginning of `malloc.c` for details.

Quicksort is internally implemented

Perl now contains its own highly optimized *qsort()* routine. The new *qsort()* is resistant to inconsistent comparison functions, so Perl's `sort()` will not provoke coredumps any more when given poorly written sort subroutines. (Some C library `qsort()`s that were being used before used to have this problem.) In our testing, the new `qsort()` required the minimal number of pair-wise compares on average, among all known `qsort()` implementations.

See `perlfunc/sort`.

Reliable signals

Perl's signal handling is susceptible to random crashes, because signals arrive asynchronously, and the Perl runtime is not reentrant at arbitrary times.

However, one experimental implementation of reliable signals is available when threads are enabled. See `Thread::Signal` Also see *INSTALL* for how to build a Perl capable of threads.

Reliable stack pointers

The internals now reallocate the perl stack only at predictable times. In particular, magic calls never trigger reallocations of the stack, because all reentrancy of the runtime is handled using a ''stack of stacks''. This should improve reliability of cached stack pointers in the internals and in XSUBs.

More generous treatment of carriage returns

Perl used to complain if it encountered literal carriage returns in scripts. Now they are mostly treated like whitespace within program text. Inside string literals and here documents, literal carriage returns are ignored if they occur paired with linefeeds, or get interpreted as whitespace if they stand alone. This behavior means that literal carriage returns in files should be avoided. You can get the older, more compatible (but less generous) behavior by defining the preprocessor symbol `PERL_STRICT_CR` when building perl. Of course, all this has nothing whatever to do with how escapes like `\r` are handled within strings.

Note that this doesn't somehow magically allow you to keep all text files in DOS format. The generous treatment only applies to files that perl itself parses. If your C compiler doesn't allow carriage returns in files, you may still be unable to build modules that need a C compiler.

Memory leaks

`substr`, `pos` and `vec` don't leak memory anymore when used in lvalue context. Many small leaks that impacted applications that embed multiple interpreters have been fixed.

Better support for multiple interpreters

The build-time option `-DMULTIPLICITY` has had many of the details reworked. Some previously global variables that should have been per-interpreter now are. With care, this allows interpreters to call each other. See the `PerlInterp` extension on CPAN.

Behavior of *local()* on array and hash elements is now well-defined

See ''Temporary Values via *local()*'' in perlsub.

`%!` **is transparently tied to the Errno module**

See perlvar(1), and Errno.

Pseudo-hashes are supported

See perlref.

`EXPR foreach EXPR` **is supported**

See perlsyn.

Keywords can be globally overridden

See perlsub.

`$^E` **is meaningful on Win32**

See perlvar.

`foreach (1..1000000)` **optimized**

`foreach (1..1000000)` is no w optimized into a counting loop. It does not try to allocate a 1000000-size list anymore.

`Foo::` **can be used as implicitly quoted package name**

Barewords caused unintuitive behavior when a subroutine with the same name as a package happened to be defined. Thus, `new Foo @args`, use the result of the call to `Foo()` instead of `Foo` being treated as a literal. The recommended way to write barewords in the indirect object slot is `new Foo:: @args`. Note that the method `new()` is called with a first argument of `Foo`, not `Foo::` when you do that.

`exists $Foo::{Bar::}` **tests existence of a package**

It was impossible to test for the existence of a package without actually creating it before. Now `exists $Foo::{Bar::}` can be used to test if the `Foo::Bar` namespace has been created.

Better locale support

See perllocale.

Experimental support for 64−bit platforms

Perl5 has always had 64-bit support on systems with 64-bit longs. Starting with 5.005, the beginnings of experimental support for systems with 32-bit long and 64-bit 'long long' integers has been added. If you add -DUSE_LONG_LONG to your ccflags in config.sh (or manually define it in perl.h) then perl will be built with 'long long' support. There will be many compiler warnings, and the resultant perl may not work on all systems. There are many other issues related to third-party extensions and libraries. This option exists to allow people to work on those issues.

prototype() **returns useful results on builtins**

See ''prototype'' in perlfunc.

Extended support for exception handling

`die()` now accepts a reference value, and `$@` gets set to that value in exception traps. This makes it possible to propagate exception objects. This is an undocumented **experimental** feature.

Re-blessing in *DESTROY()* supported for chaining *DESTROY()* methods
See ‘‘Destructors’’ in perlobj.

All `printf` format conversions are handled internally
See ‘‘printf’’ in perlfunc.

New `INIT` keyword
`INIT` subs are like `BEGIN` and `END`, but they get run just before the perl runtime begins execution. e.g., the Perl Compiler makes use of `INIT` blocks to initialize and resolve pointers to XSUBs.

New `lock` keyword
The `lock` keyword is the fundamental synchronization primitive in threaded perl. When threads are not enabled, it is currently a noop.

To minimize impact on source compatibility this keyword is ‘‘weak’’, i.e., any user-defined subroutine of the same name overrides it, unless a `use Thread` has been seen.

New `qr//` operator
The `qr//` operator, which is syntactically similar to the other quote-like operators, is used to create precompiled regular expressions. This compiled form can now be explicitly passed around in variables, and interpolated in other regular expressions. See perlop.

`our` is now a reserved word
Calling a subroutine with the name `our` will now provoke a warning when using the `-w` switch.

Tied arrays are now fully supported
See Tie::Array.

Tied handles support is better
Several missing hooks have been added. There is also a new base class for TIEARRAY implementations. See Tie::Array.

4th argument to substr
substr() can now both return and replace in one operation. The optional 4th argument is the replacement string. See ‘‘substr’’ in perlfunc.

Negative LENGTH argument to splice
splice() with a negative LENGTH argument now work similar to what the LENGTH did for *substr()*. Previously a negative LENGTH was treated as 0. See ‘‘splice’’ in perlfunc.

Magic lvalues are now more magical
When you say something like `substr($x, 5) = "hi"`, the scalar returned by *substr()* is special, in that any modifications to it affect `$x`. (This is called a ’magic lvalue’ because an ’lvalue’ is something on the left side of an assignment.) Normally, this is exactly what you would expect to happen, but Perl uses the same magic if you use *substr()*, *pos()*, or *vec()* in a context where they might be modified, like taking a reference with `\` or as an argument to a sub that modifies `@_`. In previous versions, this ’magic’ only went one way, but now changes to the scalar the magic refers to ($x in the above example) affect the magic lvalue too. For instance, this code now acts differently:

```
$x = "hello";
sub printit {
$x = "g'bye";
print $_[0], "\n";
}
printit(substr($x, 0, 5));
```

In previous versions, this would print ‘‘hello’’, but it now prints ‘‘g’bye’’.

<> now reads in records
If `$/` is a reference to an integer, or a scalar that holds an integer, <> will read in records instead of lines. For more info, see ‘‘$/’’ in perlvar.

Supported Platforms

Configure has many incremental improvements. Site-wide policy for building perl can now be made persistent, via Policy.sh. Configure also records the command-line arguments used in *config.sh*.

New Platforms

BeOS is now supported. See *README.beos*.

DOS is now supported under the DJGPP tools. See *README.dos* (installed as perldos(1) on some systems).

MiNT is now supported. See *README.mint*.

MPE/iX is now supported. See README.mpeix.

MVS (aka OS390, aka Open Edition) is now supported. See *README.os390* (installed as perlos390(1) on some systems).

Stratus VOS is now supported. See *README.vos*.

Changes in existing support

Win32 support has been vastly enhanced. Support for Perl Object, a C++ encapsulation of Perl. GCC and EGCS are now supported on Win32. See *README.win32*, aka perlwin32.

VMS configuration system has been rewritten. See *README.vms* (installed as *README_vms* on some systems).

The hints files for most Unix platforms have seen incremental improvements.

Modules and Pragmata

New Modules

B Perl compiler and tools. See B.

Data::Dumper
: A module to pretty print Perl data. See Data::Dumper.

Dumpvalue
: A module to dump perl values to the screen. See Dumpvalue.

Errno
: A module to look up errors more conveniently. See Errno.

File::Spec
: A portable API for file operations.

ExtUtils::Installed
: Query and manage installed modules.

ExtUtils::Packlist
: Manipulate .packlist files.

Fatal
: Make functions/builtins succeed or die.

IPC::SysV
: Constants and other support infrastructure for System V IPC operations in perl.

Test
: A framework for writing test suites.

Tie::Array
: Base class for tied arrays.

Tie::Handle
: Base class for tied handles.

Thread
: Perl thread creation, manipulation, and support.

attrs
: Set subroutine attributes.

fields
: Compile-time class fields.

re
: Various pragmata to control behavior of regular expressions.

Changes in existing modules

Benchmark
: You can now run tests for *x* seconds instead of guessing the right number of tests to run.

  Keeps better time.

Carp
: Carp has a new function *cluck()*. *cluck()* warns, like *carp()*, but also adds a stack backtrace to the error message, like *confess()*.

CGI
: CGI has been updated to version 2.42.

Fcntl
: More Fcntl constants added: F_SETLK64, F_SETLKW64, O_LARGEFILE for large (more than 4G) file access (the 64-bit support is not yet working, though, so no need to get overly excited), Free/Net/OpenBSD locking behaviour flags F_FLOCK, F_POSIX, Linux F_SHLCK, and O_ACCMODE: the mask of O_RDONLY, O_WRONLY, and O_RDWR.

Math::Complex
: The accessors methods Re, Im, arg, abs, rho, theta, methods can ($z->*Re()*) now also act as mutators ($z->*Re*(3)).

Math::Trig
: A little bit of radial trigonometry (cylindrical and spherical) added, for example the great circle distance.

POSIX
: POSIX now has its own platform-specific hints files.

DB_File
: DB_File supports version 2.x of Berkeley DB. See`ext/DB_File/Changes`.

MakeMaker
: MakeMaker now supports writing empty makefiles, provides a way to specify that site *umask()* policy should be honored. There is also better support for manipulation of .packlist files, and getting information about installed modules.

  Extensions that have both architecture-dependent and architecture-independent files are now always installed completely in the architecture-dependent locations. Previously, the shareable parts were shared both across architectures and across perl versions and were therefore liable to be overwritten with newer versions that might have subtle incompatibilities.

CPAN
: See perlmodinstall(1) and CPAN.

Cwd
: Cwd::cwd is faster on most platforms.

Utility Changes

`h2ph` and related utilities have been vastly overhauled.

`perlcc` a new experimental front end for the compiler is available.

The crude GNU `configure` emulator is now called `configure.gnu` to avoid trampling on `Configure` under case-insensitive filesystems.

`perldoc(1)` used to be rather slow. The slower features are now optional. In particular, case-insensitive

searches need the `-i` switch, and recursive searches need `-r`. You can set these switches in the `PERLDOC` environment variable to get the old behavior.

Documentation Changes

Config.pm now has a glossary of variables.

Porting/patching.pod has detailed instructions on how to create and submit patches for perl.

perlport(1) specifies guidelines on how to write portably.

perlmodinstall(1) describes how to fetch and install modules from `CPAN` sites.

Some more Perl traps are documented now. See perltrap.

perlopentut(1) gives a tutorial on using *open()*.

perlreftut(1) gives a tutorial on references.

perlthrtut(1) gives a tutorial on threads.

New Diagnostics

Ambiguous call resolved as CORE::%s(), qualify as such or use &

(W) A subroutine you have declared has the same name as a Perl keyword, and you have used the name without qualification for calling one or the other. Perl decided to call the builtin because the subroutine is not imported.

To force interpretation as a subroutine call, either put an ampersand before the subroutine name, or qualify the name with its package. Alternatively, you can import the subroutine (or pretend that it's imported with the `use subs` pragma).

To silently interpret it as the Perl operator, use the `CORE::` prefix on the operator (e.g. `CORE::log($x)`) or by declaring the subroutine to be an object method (see ''attrs'').

Bad index while coercing array into hash

(F) The index looked up in the hash found as the 0'th element of a pseudo-hash is not legal. Index values must be at 1 or greater. See perlref.

Bareword ''%s'' refers to nonexistent package

(W) You used a qualified bareword of the form `Foo::`, but the compiler saw no other uses of that namespace before that point. Perhaps you need to predeclare a package?

Can't call method ''%s'' on an undefined value

(F) You used the syntax of a method call, but the slot filled by the object reference or package name contains an undefined value. Something like this will reproduce the error:

```
$BADREF = 42;
process $BADREF 1,2,3;
$BADREF->process(1,2,3);
```

Can't check filesystem of script ''%s'' for nosuid

(P) For some reason you can't check the filesystem of the script for nosuid.

Can't coerce array into hash

(F) You used an array where a hash was expected, but the array has no information on how to map from keys to array indices. You can do that only with arrays that have a hash reference at index 0.

Can't goto subroutine from an eval-string

(F) The ''goto subroutine'' call can't be used to jump out of an eval ''string''. (You can use it to jump out of an eval {BLOCK}, but you probably don't want to.)

Can't localize pseudo-hash element

(F) You said something like `local $ar->{'key'}`, where `$ar` is a reference to a pseudo-hash. That hasn't been implemented yet, but you can get a similar effect by localizing the corresponding array element directly: `local $ar->[$ar->[0]{'key'}]`.

Can't use %%! because Errno.pm is not available
(F) The first time the %! hash is used, perl automatically loads the Errno.pm module. The Errno module is expected to tie the %! hash to provide symbolic names for `$!` errno values.

Cannot find an opnumber for "%s"
(F) A string of a form `CORE::word` was given to *prototype()*, but there is no builtin with the name `word`.

Character class syntax [. .] is reserved for future extensions
(W) Within regular expression character classes ([]) the syntax beginning with "[." and ending with ".]" is reserved for future extensions. If you need to represent those character sequences inside a regular expression character class, just quote the square brackets with the backslash: "\[." and ".\]".

Character class syntax [: :] is reserved for future extensions
(W) Within regular expression character classes ([]) the syntax beginning with "[:" and ending with ":]" is reserved for future extensions. If you need to represent those character sequences inside a regular expression character class, just quote the square brackets with the backslash: "\[:" and ":\]".

Character class syntax [= =] is reserved for future extensions
(W) Within regular expression character classes ([]) the syntax beginning with "[=" and ending with "=]" is reserved for future extensions. If you need to represent those character sequences inside a regular expression character class, just quote the square brackets with the backslash: "\[=" and "=\]".

`%s`: Eval-group in insecure regular expression
(F) Perl detected tainted data when trying to compile a regular expression that contains the `(?{ ... })` zero-width assertion, which is unsafe. See "(?{ code })" in perlre(1), and perlsec.

`%s`: Eval-group not allowed, use re 'eval'
(F) A regular expression contained the `(?{ ... })` zero-width assertion, but that construct is only allowed when the `use re 'eval'` pragma is in effect. See "(?{ code })" in perlre.

`%s`: Eval-group not allowed at run time
(F) Perl tried to compile a regular expression containing the `(?{ ... })` zero-width assertion at run time, as it would when the pattern contains interpolated values. Since that is a security risk, it is not allowed. If you insist, you may still do this by explicitly building the pattern from an interpolated string at run time and using that in an *eval()*. See "(?{ code })" in perlre.

Explicit blessing to '' (assuming package main)
(W) You are blessing a reference to a zero length string. This has the effect of blessing the reference into the package main. This is usually not what you want. Consider providing a default target package, e.g. bless($ref, `$p` || 'MyPackage');

Illegal hex digit ignored
(W) You may have tried to use a character other than 0 - 9 or A - F in a hexadecimal number. Interpretation of the hexadecimal number stopped before the illegal character.

No such array field
(F) You tried to access an array as a hash, but the field name used is not defined. The hash at index 0 should map all valid field names to array indices for that to work.

No such field "%s" in variable `%s` of type `%s`
(F) You tried to access a field of a typed variable where the type does not know about the field name. The field names are looked up in the `%FIELDS` hash in the type package at compile time. The `%FIELDS` hash is usually set up with the 'fields' pragma.

Out of memory during ridiculously large request
(F) You can't allocate more than 2^31+"small amount" bytes. This error is most likely to be caused by a typo in the Perl program. e.g., `$arr[time]` instead of `$arr[$time]`.

Range iterator outside integer range
(F) One (or both) of the numeric arguments to the range operator ".." are outside the range which can be represented by integers internally. One possible workaround is to force Perl to use magical string

increment by prepending ‘‘0’’ to your numbers.

Recursive inheritance detected while looking for method ’%s’ %s
(F) More than 100 levels of inheritance were encountered while invoking a method. Probably indicates an unintended loop in your inheritance hierarchy.

Reference found where even-sized list expected
(W) You gave a single reference where Perl was expecting a list with an even number of elements (for assignment to a hash). This usually means that you used the anon hash constructor when you meant to use parens. In any case, a hash requires key/value **pairs**.

```
%hash = { one => 1, two => 2, }; # WRONG
%hash = [ qw/ an anon array / ]; # WRONG
%hash = ( one => 1, two => 2, ); # right
%hash = qw( one 1 two 2 ); # also fine
```

Undefined value assigned to typeglob
(W) An undefined value was assigned to a typeglob, a la `*foo = undef`. This does nothing. It’s possible that you really mean `undef *foo`.

Use of reserved word ‘‘%s’’ is deprecated
(D) The indicated bareword is a reserved word. Future versions of perl may use it as a keyword, so you’re better off either explicitly quoting the word in a manner appropriate for its context of use, or using a different name altogether. The warning can be suppressed for subroutine names by either adding a & prefix, or using a package qualifier, e.g. `&our()`, or `Foo::our()`.

perl: warning: Setting locale failed.
(S) The whole warning message will look something like:

```
perl: warning: Setting locale failed.
perl: warning: Please check that your locale settings:
LC_ALL = "En_US",
LANG = (unset)
are supported and installed on your system.
perl: warning: Falling back to the standard locale ("C").
```

Exactly what were the failed locale settings varies. In the above the settings were that the LC_ALL was ‘‘En_US’’ and the LANG had no value. This error means that Perl detected that you and/or your system administrator have set up the so-called variable system but Perl could not use those settings. This was not dead serious, fortunately: there is a ‘‘default locale’’ called ‘‘C’’ that Perl can and will use, the script will be run. Before you really fix the problem, however, you will get the same error message each time you run Perl. How to really fix the problem can be found in ‘‘LOCALE PROBLEMS’’ in perllocale.

Obsolete Diagnostics

Can’t *mktemp()*
(F) The *mktemp()* routine failed for some reason while trying to process a **-e** switch. Maybe your /tmp partition is full, or clobbered.

Removed because **-e** doesn’t use temporary files any more.

Can’t write to temp file for **-e**: %s
(F) The write routine failed for some reason while trying to process a **-e** switch. Maybe your /tmp partition is full, or clobbered.

Removed because **-e** doesn’t use temporary files any more.

Cannot open temporary file
(F) The create routine failed for some reason while trying to process a **-e** switch. Maybe your /tmp partition is full, or clobbered.

Removed because **-e** doesn’t use temporary files any more.

regexp too big

(F) The current implementation of regular expressions uses shorts as address offsets within a string. Unfortunately this means that if the regular expression compiles to longer than 32767, it'll blow up. Usually when you want a regular expression this big, there is a better way to do it with multiple statements. See perlre.

Configuration Changes

You can use ''Configure -Uinstallusrbinperl'' which causes installperl to skip installing perl also as /usr/bin/perl. This is useful if you prefer not to modify /usr/bin for some reason or another but harmful because many scripts assume to find Perl in /usr/bin/perl.

BUGS

If you find what you think is a bug, you might check the headers of recently posted articles in the comp.lang.perl.misc newsgroup. There may also be information at http://www.perl.com/perl/ , the Perl Home Page.

If you believe you have an unreported bug, please run the **perlbug(1)** program included with your release. Make sure you 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.com*> to be analysed by the Perl porting team.

SEE ALSO

The *Changes* file 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.

HISTORY

Written by Gurusamy Sarathy <*gsar@activestate.com*>, with many contributions from The Perl Porters.

Send omissions or corrections to <*perlbug@perl.com*>.