Renée Bäcker

Was ist neu in Perl 5.18?

Noch in diesem Monat soll eine neue Version von Perl5 veröffentlicht werden: Perl 5.18. Auch mit dieser Version wird es einige Neuerungen geben, von denen die wichtigsten in diesem Artikel vorgestellt werden.

Wie schon in den Vorgänger-Versionen sind es eher kleinere Änderungen und nicht der ganz große Wurf an neuen Features. Schon in der vergangenen Ausgabe von *$foo* wurde der neue Hashing-Algorithmus vorgestellt. Deshalb an dieser Stelle nur der Hinweis, dass man sich jetzt auch auf ein und demselben Rechner nicht mehr auf die Reihenfolge von Schlüsseln in einem Hash verlassen kann.

```
test.pl

use Data::Dumper;

my %h = (1..6);
print Dumper \%h;
```

Unter bisherigen Perl-Versionen war die Reihenfolge eines Hashs bei jedem Lauf gleich - wobei die Reihenfolge bei jeder Maschine anders war.

```
$ perlbrew switch perl-5.16.2
$ perl test.pl
$VAR1 = {
          '1' => 2,
          '3' => 4,
          '5' => 6
        };
$ perl test.pl
$VAR1 = {
          '1' => 2,
          '3' => 4,
          '5' => 6
        };
```

Unter Perl 5.18 (und den letzten Developer-Releases) ist die Reihenfolge bei jedem Programmlauf unterschiedlich.

```
$ perlbrew switch perl-5.17.10
$ perl test.pl
$VAR1 = {
          '3' => 4,
          '5' => 6,
          '1' => 2
        };
$ perl test.pl
$VAR1 = {
          '5' => 6,
          '1' => 2,
          '3' => 4
        };
```

Da es zu einiger Unsicherheit bei Programmierern kam, noch ein kurzer Hinweis zu `keys` und `values`: Wenn man `keys` aufruft kann man auch weiterhin sicher sein, dass `values` die dazu passenden Werte in der richtigen Reihenfolge liefert.

```
$ perl test.pl
$VAR1 = {
          '5' => 6,
          '1' => 2,
          '3' => 4
        };
keys: 5, 1, 3
values: 6, 2, 4
```

Änderungen bei Regulären Ausdrücken

Spezialvariablen `$&`, `$'` und `` $` `` sind nicht länger langsam. Bisher hat die Verwendung dieser Spezialvariablen das komplette Programm ausgebremst (siehe auch $foo Ausgabe Nr. 4), und man hat die verwendeten Module nicht unter Kontrolle. Mit Perl 5.18 sind diese Variablen keine Performancefresser mehr.

Diese Spezialvariablen können verwendet werden, um den gematchten Teilstring, den Teil vor dem Match bzw. nach dem Match herauszubekommen:

```
$ perl -E 'q~$foo - DAS Magazin zur
   Softwareentwicklung mit Perl~ =~
    m{Magazin}; say $&; say $`'
Magazin
$foo - DAS
```

Einfache ungeschützte geschweifte Klammern sind jetzt als *deprecated* markiert. Geschweifte Klammern müssen jetzt entweder mit einem Backslash geschützt werden oder Teil eines Ausdrucks sein, in dem die Klammer ein Metazeichen ist.

Erlaubt:

```
/test{2,4}/
/test\{hallo\}/
```

Nicht mehr erlaubt:

```
/test{,4}/
```

Whitespaces matcht man mit `\s`. Dazu zählen jetzt auch die vertikalen Tabs.

In Regulären Ausdrücken kann man Perl-Code einbetten. Dazu gibt es die Konstrukte `(?{})` und `(??{})`. Zwar bleibt dieses Feature als experimentell gekennzeichnet, aber Dave Mitchell hat in den letzten Monaten das komplette Feature überarbeitet und dabei etliche Fehler behoben.

Die Code-Blöcke werden zur gleichen Zeit geparst wie der umgebende Code.

Ein Beispiel für eingebetteten Perl-Code in Regulären Ausdrücken:

```
$_ = 'a' x 8;
m<
   (?{ $cnt = 0 })        # Initialize $cnt.
   (
     a
     (?{
       local $cnt = $cnt + 1; # Update $cnt,
                         # backtracking-safe.
     })
   )*
   aaaa
   (?{ $res = $cnt })    # On success copy to
                     # non-localized location
>x;
```

Ein neues Feature ist die Einführung von Set-Operationen in Regulären Ausdrücken. Damit kann man Zeichenklassen zusammen führen, einzelne Zeichen aus einer großen Zeichenklasse herausnehmen oder Alternativen einführen. Sollen z.B. alle Großbuchstaben von A bis Z gematcht werden mit den Ausnahmen C, K bis N und Y, dann kann man das bisher so schreiben:

```
/[ABD-JO-XZ]/
```

Mit den neuen Sets ist folgendes möglich

```
/(?[ [A-Z] - [CK-NY] ])/
```

Das wirft unter Perl 5.18 noch eine Warnung *The regex_sets feature is experimental in regex;...*, die man mit `no warnings "experimental::regex_sets"` ausschalten kann.

Lexikalische Subroutinen

Ein lang gehegter Wunsch vieler Programmierer wird jetzt Wirklichkeit: Lexikalische Subroutinen. Es ist zwar noch als experimentell gekennzeichnet, aber es ist schon ganz brauchbar. Um damit arbeiten zu können, muss man folgende drei Zeilen schreiben:

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

Jetzt kann man folgendes schreiben:

```
package TestLexicalSubs;
{
  use 5.018;
  no warnings "experimental::lexical_subs";
  use feature "lexical_subs";

  my sub test;
  sub test {
    say "test";
  }

  sub hallo {
    say "hallo";
    test();
  }
}

package main;

use feature 'say';

TestLexicalSubs::hallo();
eval{
  TestLexicalSubs::test()
} or say "Fehler: $@";
```

Das erzeugt folgende Ausgabe:

```
hallo
test
Fehler: Undefined subroutine
  &TestLexicalSubs::test called ...
```

Wie man sieht, kann man auf die Subroutine `test` nur innerhalb des Blocks in dem sie deklariert wurde aufrufen.

Neben `my sub ...` kann man auch `our sub ...` und `state sub ...` verwenden. In der Dokumentation `perlsub` findet man einen größeren Abschnitt zu lexikalischen Subroutinen.

Sonstiges

`qw()` kann nicht mehr als Klammernpaar bei `for`-Schleifen verwendet werden. An vielen Codestellen konnte man `qw()` als Ersatz für runde Klammern finden. Z.B.

```
for my $name qw(hannah martin paula) {
    say $name;
}
```

Schon mit Perl 5.14 wurde das als *deprecated* markiert. Mit Perl 5.18 bricht das Programm ab:

```
syntax error at -e line 1, near
    "$name qw(hannah martin paula)"
Execution of -e aborted due to
    compilation errors.
```

In nachgestellten Konstrukten funktioniert das aber weiterhin:

```
$ perl -E 'say for qw(hannah martin paula)'
hannah
martin
paula
```

Neuer Mechanismus für experimentelle Features: Wie schon in einigen Beispielen in diesem Artikel gezeigt, gibt es einen neuen Mechanismus für experimentelle Features. Der `use feature`-Aufruf bleibt wie bisher, allerdings geben die (meisten) experimentellen Features eine Warnung aus, die mittels `no warnings "experimental::<feature_name>"` deaktiviert werden können.

Perl 5.18 wird mit Unicode 6.2 ausgeliefert.

Wenn ein Zeichen, das mittels `\N{...}` benannt wird, nicht existiert, wird jetzt ein Fehler geworfen. Eigene Aliase (siehe *CUSTOM ALIASES* in `charnames`) können jetzt auch Nicht-Latin1-Zeichen enthalten. Somit wäre auch ein `\N{þ€ł}` möglich.

Kein neues Feature, aber eine Änderung an einem bestehenden Feature, ist das Markieren der *switch*-Familie als experimentell. Ein anderes Feature, das mit Perl 5.10 eingeführt wurde, ist jetzt als *deprecated* markiert: Das lexikalische `$_`. Es gab damit einige Probleme: Einige CPAN-Module erwarten, dass sie mit dem globalen `$_` arbeiten (z.B. `List::Util`). Prototypen haben damit Probleme gehabt und selbst XS-Code konnte nicht darauf zugreifen.

Seit Perl 5.10 wurde `CPANPLUS` als alternativer CPAN-Client mitgeliefert. Nachdem einige Probleme mit `CPAN.pm` unter VMS beseitigt wurden, wird `CPANPLUS` nicht mehr mit ausgeliefert.

In bisherigen Perl-Versionen wurden alle Kategorien an Warnungen ausgeschaltet, wenn nur eine bestimmte Kategorie aktiviert wurde:

```
use warnings;

print $*; # "deprecated" Warnung

use warnings "void";
print $*; # keine Warnung
```

Ab Perl 5.18 bleiben einige Kategorien aktiv, bis explizit `no warnings` aufgerufen wurde. Dazu zählen auch *deprecated*-Warnungen:

```
use warnings;

print $*; # "deprecated" Warnung

use warnings "void";
print $*; # keine Warnung
```

Dieses Beispiel gibt die Warnung

```
$* is no longer supported, and will
   become a syntax error at - line 3.
$* is no longer supported, and will become
   a syntax error at - line 7.
```

aus.

Wenn beim Einlesen von Dateien der Variablen `$/` eine Re-

ferenz auf eine Zahl zugewiesen wurde, wurde mit <> diese Anzahl von Bytes gelesen:

```
$/ = \500;
open my $fh, '<', $file or die $!;
my $five_hundred_bytes = <$fh>;
```

Das kann Probleme machen wenn die Datei UTF-8-Daten enthält und diese auch mit dem `:encoding`-Perl I/O-Layer geöffnet wurde. Dann konnte es nämlich passieren, dass bei einem Zwei-Byte-Zeichen nur die Hälfte des Zeichens eingelesen wurde. Ab Perl 5.18 werden bei der Verwendung des `:encoding`-Perl I/O-Layer nicht *n* Bytes, sondern *n* Zeichen gelesen:

```
$/ = \500;
open my $fh, '<:encoding(utf-8)', $file
    or die $!;
my $five_hundred_chars = <$fh>;
```

Geschwindigkeitsverbesserungen

Auch an der Geschwindigkeit wurde gearbeitet. So gibt es einige signifikante Verbesserungen:

Wenn eine Kopie eines Strings, auf den ein Match ausgeführt wird, notwendig ist, wird nur der Teilstring kopiert, der für die Capture-Variable notwendig ist. Bisher wurde immer der komplette String kopiert, was bei sehr großen Strings viel Speicher und Zeit benötigt hat.

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

profitiert sehr stark von dieser Optimierung.

Bei Regulären Ausdrücken ist das Matchen von Unicodezeichen schneller geworden. Der größte Sprung wurde bei der Verwendung von `\X` erreicht, bei dem ein Geschwindigkeitsvorteil von bis zu 40% erreicht wird. Auch Zeichenklassen mit Zeichen, deren Codepoint über 255 liegt, sind jetzt schneller.

Bei der Verwendung von *NO_TAINT_SUPPORT* bei der Kompilierung von Perl wird der Taint-Modus von Perl komplett abgeschaltet. Das sollte man allerdings nur benutzen, wenn man genau weiß, wofür das Perl verwendet wird. Man sollte dabei auch bedenken, dass dann die Tests einiger CPAN-Module und auch des Perl-Kerns selbst fehlschlagen. Allerdings bekommt man einen kleinen Geschwindigkeitsvorteil.

Sicherheit

Neben dem Rehashing, das oben beschrieben wurde, gibt es noch weitere Sicherheitsfixes und die Dokumentation wurde teilweise angepasst:

Wenn es Benutzern erlaubt ist, beliebige Faktoren für `x` zu bestimmen (z.B. `'string' x $userinput`), kann das für einem DoS-Angriff genutzt werden. In Perl < 5.15 konnte es sogar für einen Pufferüberlauf genutzt werden. Dieses Problem ist jetzt behoben.