Renée Bäcker

Neuerungen in Perl 5.14

Perl 5.14 ist zwar noch nicht erschienen, aber im April wird es soweit sein. Seit Perl 5.12.0 veröffentlicht wurde, waren die Perl 5 Porters und andere Entwickler sehr fleißig und haben einige Neuerungen in Perl 5.14 untergebracht. Die hier gezeigten Änderungen spiegeln den Stand von Perl 5.13.8 (Dezember 2010) wider und bringen Lust auf mehr.

Unicode-Unterstützung

Unicode ist eigentlich immer ein Thema - auch eines, das immer wichtiger wird. Seit einiger Zeit arbeitet vor allem Karl Williamson an der verbesserten Unicode-Unterstützung in Perl. Zum einen wurde auf die Version 6.0 von Unicode aktualisiert. Auch bei den Regulären Ausdrücken hat sich in Bezug auf Unicode einiges getan.

Mit der neuen Unicode-Version sind 2.088 neue Zeichen aufgenommen werden. Im Gegensatz zu früheren Perl-Versionen kennt `charnames` jetzt auch alle Zeichen aus der Unicode-Datenbank.

Unicode-Zeichen können einfach mittels `\N{name}` verwendet werden:

```
my $string = "I \N{HEAVY BLACK HEART} Perl";
```

An diesen `\N{...}`-Werten gab es auch Änderungen: So ist `\N{BELL}` jetzt als `deprecated` markiert, weil Unicode diesen Namen für ein anderes Zeichen verwendet. Außerdem können auch bestimmte Abkürzungen statt der langen Namen verwendet werden:

```
my $string = "\N{NBSP}";
# ist das gleiche wie
my $string = "\N{NO-BREAK SPACE}";
```

Für Reguläre Ausdrücke gibt es drei neue Flags: `u`, `d` und `l`.

Wird das Flag `l` verwendet, wird für die Regulären Ausdrücke die `locale`-Einstellung berücksichtigt. Bei dem folgenden Programm wird im ersten Fall nichts ausgegeben, weil bei `en_US.UTF-8` das "ä" nicht als Buchstabe vorkommt. Nachdem das locale auf `de_DE.ISO-8859-1` geändert wird, wird ausgegeben, dass der Reguläre Ausdruck matcht. Im ISO-8859-1 Zeichensatz ist das "ä" als Buchstabe enthalten.

```
use charnames qw(:full);
use POSIX qw(locale_h);

setlocale( LC_CTYPE, 'en_US.UTF-8' );

my $text = "ä";

{
    use locale;
    print $text =~ /(?l)\w/;
}

setlocale( LC_CTYPE, 'de_DE.ISO-8859-1' );

{
    use locale;
    print $text =~ /(?l)\w/;
}
```

Mit dem Flag `u` matcht z.B. `\w` alle Unicode-Zeichen, die als "Letter" bekannt sind.

```
my $text = "ä";
print $text =~ /(?u)\w/;
```

Möchte man das `u`-Flag für alle Regulären Ausdrücke in einem Scope einschalten, kann man das auch mit `use feature 'unicode_strings'` machen.

```
use feature 'unicode_strings';

my $text = "ä";
print $text =~ /\w/;
```

Das Flag `d` erzwingt das "alte" Verhalten von Perl - wie es vor Perl 5.14 war. Was `\w` bedeutet, hängt hier also vom String ab, der durchsucht werden soll. Also auch in solchen Fällen, in denen `use feature 'unicode_strings'` aktiviert ist.

```
use feature 'unicode_strings';

my $text = "ä";
print $text =~ /\w/;
print $text =~ /(?d)\w/;
```

Bei diesem Beispiel matcht der erste Reguläre Ausdruck, da durch das 'unicode_strings' die neue Unicode-Semantik für Reguläre Ausdrücke aktiviert wird. Der zweite Ausdruck matcht aber nicht, weil durch das `d`-Flag das alte Verhalten wieder eingeschaltet wird.

Zu beachten ist hierbei, dass die Regex-Engine **immer** die Unicode-Semantik nimmt, sobald für den String das UTF-8-Flag angeschaltet ist. Dann hat auch das `l`-Flag keine Auswirkung.

Karl Williamson war so fleißig, dass die Menge der Änderungen hier nicht reinpasst. Insgesamt wurde die Unicode-Unterstützung aber stark verbessert. Als weiterführende Literatur empfehle ich diese `perldoc`s:

- perlunitut
- perlunifaq
- perlunicode
- perluniintro
- Encode
- charnames

Moritz Lenz hatte für die 5. Ausgabe von $foo einen ausführlichen Artikel zu Charsets und Unicode geschrieben.

Reguläre Ausdrücke

Bei den Regulären Ausdrücken gibt es noch weitere Änderungen:

Wer mit stringifizierten Regulären Ausdrücken arbeitet, muss mit Perl 5.14 ein paar Änderungen vornehmen. Bisher war es so, dass ein stringifizierter RegEx mit `(?...)` begann. Das musste durch die Erweiterungen, die im Zuge der Entwicklungen hinzugekommen sind, angepasst werden. Jetzt beginnt der String mit `(?^...)`.

```
my $regex = qr/[0-9]/;
print "$regex";
```

Unter Perl 5.12 wird `(?-xism:[0-9])` ausgegeben, unter Perl 5.14 ist die Ausgabe `(?^:[0-9]`.

Die Modifikatoren der Regulären Ausdrücke können jetzt auch über `user re ...` für einen Scope "global" aktiviert werden, ohne sie bei jedem Regulären Ausdruck hinschreiben zu müssen.

```
my $regex = qr{ \A \d+ };
print "1: $regex\n";

{
    use re '/xms';
    my $regex = qr{ \A \d+ };
    print "2: $regex\n";
}
```

Liefert als Ausgabe:

```
1: (?^: \A \d+)
2: (?^xms: \A \d+)
```

Änderungen bei Operatoren und Funktionen

Ein sehr nützliches Feature ist die Erweiterung der Operatoren `s///` und `y///`. Bisher hatten diese Operatoren die als Rückgabewert entweder Erfolg/Misserfolg (1/`undef`) bzw. die Anzahl der Ersetzungen. Durch den neuen Modifikator `r` wird jetzt der veränderte Wert zurückgeliefert und der Variableninhalt bleibt erhalten. Bisher musste man Code so schreiben:

```
(my $neu = $alt) =~ s/alt/neu/;
my @neue_werte =
   map{ s/alt/neu/; $_ }@alte_werte;
```

Unter Perl 5.14 kann man das besser schreiben als:

```
my $neu = $alt =~ s/alt/neu/r;
my @neue_werte =
   map{ s/alt/neu/r }@alte_werte;
```

Funktionen, die mit Arrays und Hashes arbeiten können mit der neuen Perl-Version auch mit den entsprechenden Referenzen umgehen. Sollen Werte zu einer Arrayreferenz hinzugefügt werden, muss beim `push` die Arrayreferenz nicht mehr dereferenziert werden. Es funktioniert einfach

```
push $arrayref, $neuer_wert;
```

Die weiteren Funktionen, die jetzt auch Referenzen akzeptieren sind `push`, `shift`, `unshift`, `pop`, `splice`, `keys`, `values` und `each`.

Sonstiges

Unter Linux kann der Spezialvariablen `$0` (Name des Programms) ein neuer Wert zugewiesen werden. Dieser Wert wird dann auch bei Programmen wie `top` oder `ps` angezeigt.

```
$0 = 'mein perl programm';
```

In Perl kann man das unäre – auch auf Strings anwenden. Bisher war es so, dass bei einem "positiven" String ("ein test") einfach ein "-" vorangestellt wurde ("-ein test") und bei "negativen" Strings ("-ein test") wurde aus dem "-" ein "+" ("+ein test"). Das wurde auch gemacht, wenn der String eine Zahl war (also z.B. "1"). Bei Strings aus Zahlen hat sich das jetzt geändert und das Verhalten wurde so angepasst, dass es gleich zu dem Verhalten von normalen Zahlen ist:

```
my $var = - 'ein test';  # -ein test
my $var = - '-ein test'; # +ein test
my $var = - '1';         # -1

my $var = - '-1';        # alt: +1, neu: 1
```

Verwendet man mehrere `package`s in einer Datei muss man mit dem Scope aufpassen. Definiert man in einem `package` eine Variable, ist diese auch in den anderen `package`s in der Datei sichtbar. Möchte man das beschränken, musste man bisher

```
{
    package Foo;

    # weiterer Code
}
```

schreiben. Um den Code lesbarer zu halten, wurde hier die Möglichkeit geschaffen, das `package` vor dem Block zu definieren:

```
package Foo;
{
    # weiterer Code
}
```

Prototypen sind in Perl ein eigenes Thema. Bei Perl-Einsteigern sieht man häufiger die Verwendung von Prototypen, weil sie versuchen ihr Wissen von Methodensignaturen auf Perl zu übertragen. Dabei sind Prototypen in Perl etwas ganz anderes.

Mit Perl 5.14 gibt es ein paar Neuerungen in Bezug auf Prototypen. Methoden mit den Prototypen `(*)`, `(;$)` oder `(;*)` werden jetzt mit einer höheren Priorität versehen.

```
my $nr = 5;
sub foo(;$) { $_[0] * 2 };
print foo $nr < 3;
```

Wird jetzt als `print foo($nr) < 3` geparst.

Möchte man Funktionen schreiben, die sowohl Arrays/Hashes bzw. Referenzen auf diese Strukturen erlauben, kann man den `+`-Prototypen verwenden. Ohne diese Änderungen, müsste man sich vorher entscheiden, wie der Parameter aussehen muss.

```
use Data::Dumper;

sub test(\@)  { print Dumper \@_ }
sub test2(+@) { print Dumper \@_ }

my @array = ( 1 .. 3 );

test( @array );
test2( @array );
test2( \@array );
```

Bei der Subroutine `test` muss ein Array übergeben werden. Versucht man eine Arrayreferenz zu übergeben, bekommt man die Fehlermeldung

```
Type of arg 1 to main::test must be array
```

Der Subroutine `test2` hingegen kann man sowohl ein Array als auch eine Arrayreferenz übergeben und es kommt jeweils als Arrayreferenz in der Subroutine an.

Manchmal sieht man so etwas:

```
for my $var qw(1 2 3) {
    # ...
}
```

In solchen Fällen wurde das qw// ohne runde Klammern außenherum als `deprecated` markiert. Man muss den Code jetzt so schreiben:

```
for my $var (qw(1 2 3)) {
    # ...
}
```

Weitere Änderungen im Schnelldurchlauf: `given` hat jetzt einen Rückgabewert (man muss aber `do` verwenden); Am Errorhandling wurde einiges gemacht; Stashes sind immer definiert; Bei getieten Variablen wird bei `local` das `tie` aufgehoben.

In Perl gibt es auch Module, die als "deprecated" markiert sind, teilweise schon seit einigen Jahren. Ein paar dieser Module wurden jetzt aus dem Core entfernt. Damit Programmierer diese weiterhin nutzen können, müssen die Module vom CPAN installiert werden. Module, die entfernt wurden:

- Switch
- Class::ISA
- Pod::Plainer

Neben diesen Änderungen wurden noch sehr viele Module aktualisiert, deren Änderungen hier zu weit führen würden. Auch am Core wurden noch weitere Änderungen durchgeführt, die ich hier nicht alle aufzeigen kann. Viele haben keine Änderungen für den Programmierer zur Folge, sondern sind Optimierungen in Bezug auf Geschwindigkeit oder Speicherverbrauch.