



Serious Perl: The Absolute Minimum You Need To Know

Henning Koch
hkoch@paws.cx

Serious Perl

Perl's extremely flexible syntax makes it easy to write code that is harder to read and maintain than it could be. This column describes some very basic practices I consider necessary for a clear and concise style of writing Perl.

I wish I had known

There must be a special circle in hell reserved for people who abused Perl the way I did.

I started out using Perl for CGI scripts and simple automation tasks. Having converted to Perl from Pascal, I enjoyed the ease and expressiveness of the language, the coding was fun and I was happy. Things went downhill when the web applications I developed grew to a point where complexity began to rear its ugly head. I found it increasingly hard to keep track of what was going on in my programs. Maintaining old code became a living nightmare.

I realized that my procedural spaghetti hacks were not going to suffice anymore. I somehow had to make my code suck less. Unfortunately finding help turned out to be harder than I had thought. Every piece of Perl advice I encountered either seemed to be concerned with the very basics or led even further into the guts of Perl hackery. Surprisingly little information dealt with best practices and neither did Perl's extremely flexible syntax help to figure out the Right Thing To Do.

Two years later, I had finally made my peace with Perl. In a mood I spent an afternoon compiling my knowledge into an article which I boldly titled "Writing Serious Perl: The absolute minimum you need to know", put it on my website and asked a few friends to check it out. The amount reception it received was unexpected, to say the least, and made me realize that there were thousands of other Perl hackers who were looking for the same kind of assistance that I once was.

Instant data structures

Use {} to create anonymous hash references. Use [] to create anonymous array references. Combine these constructs to create more complex data structures such as lists of hashes:

```
my @students = (
  { name      => 'Clara',
    registration => 10405,
    grades     => [ 2, 3, 2 ]
  },
  { name      => 'Amy',
    registration => 47200,
    grades     => [ 1, 3, 1 ]
  },
  { name      => 'Deborah',
    registration => 12022,
    grades     => [ 4, 4, 4 ]
  }
);
```

Use the arrow operator -> to dereference the structure and get to the values.

```
# print out names of all students
foreach my $student (@students) {
    print $student->{name} . "\n";
}

# print out Clara's second grade
print $students[0]->{grades}->[1];

# delete Clara's registration code
delete $students[0]->{registration};
```

Namespaces

One package should never mess with the namespace of another package unless it has been explicitly told to do so. Thus, never define methods in another script and require it in. Always wrap your library in