Just do{} It

brian d foy

I recently went through a code review with a client (it was my code up for review) and had to give one of my favorite idioms: the **do{}** block. I can understand that because a lot of the people who will probably have to maintain the code might not be used to this handy little feature of Perl (which some people might call a Perly switch statement).

I got hooked on **do{}** when I had to refactor some code that was a long series of if-elsif branches that existed to set the value of a single variable. That variable name showed up a lot in the code, and I wanted it to show up only once.

```
my $pie = '';
if( $day eq 'Monday' )
   { $pie = 'Apple' }
elsif( $day eq 'Tuesday' )
   { $pie = 'Cherry' }
elsif ...
elsif( $day eq 'Sunday' )
   { $pie = 3.14 }
```

I think of do{} as an inline subroutine, and just like a subroutine it is going to return the last evaluated expression. The last evaluated expression is not necessarily the last statement in the do{} block, though.

I can refactor my example to get rid of the duplicated variable names. The last evaluated expression is the statement for whichever branch the code takes. If **$day** is Tuesday, the last evaluate expression inside the **do{}** is going to be the string Cherry, which becomes the return value. I assign that value to **$pie**. I have to remember to get the semicolon after the **do{}** because its a statement:

```
my $pie = do {
      if( $day eq 'Monday'  ) { 'Apple' }
   elsif( $day eq 'Tuesday' ) { 'Cherry' }
   elsif ...
   elsif( $day eq 'Sunday'  ) { 3.14 }
};
```

Now that I have **$pie** in only one place, if I change that variable, perhaps to be **$meal{pie}**, I have minimal changes to make.

I can do this with more than one variable too. Inside each branch I simply return a list. The first item I return ends up in **$salad**, the second in **$meat**, and the third in **$pie**.

```
my( $salad, $meat, $pie ) = do {
      if( $day eq 'Monday' )
         { qw(Caeser Steak Apple) }
   elsif( $day eq 'Tuesday' )
         { qw(Cobb Lamb Cherry) }
   ...
};
```

I can then easily change that to a hash slice assignment since I only have to make the change in one place.

```
@meal{ qw(salad meat die) } = do {...};
```

So now you know, and this still might show up in my code. ■

Here's some code I ran across to create a multi-level hash without pain and suffering. A lot of people might write a recursive function to do this (me included). A recursive function starts at the top of the hash and works its way down. This little bit starts at the bottom and works its way up.

It starts by assigning the value to the variable that will hold the hash, then created an anonymous hash from that and assigning it back to itself. Do this for each of the elements in **keys** in reverse order (because it starts from the bottom).

```
my @keys = 'a' .. 'f';
my $hash = 'value';
$hash = { $_=> $hash } for reverse @keys;
```

The end result is the multi-level hash. ■