



Hash Anti-Patterns

Alberto Manuel Simões
 ambs@di.uminho.pt

In “Array Anti-Patterns” (TPR 1.3), I showed you some bad ways of using arrays. The same can happen when using hashes, which used to be called associative arrays.

It is important to know when you need a hash. Then, it is important to know the tricks Perl has to make them really useful. The proper use of hashes reduces the complexity of your code and makes it more readable to the people who have to maintain it.

When you need them

One of the first symptoms for the use of a hash is when I use several different variables represent to different properties for the same thing. For instance, I could define two database connections as with a series of six scalar variables:

```
my $main_server_addr      = "some.url";
my $main_server_port     = 4000;
my $main_server_username = "me";

my $backup_server_addr   = "back.url";
my $backup_server_port   = 4444;
my $backup_server_username = "me";
```

This kind of code causes two main problems. First, I’m polluting the Perl variables namespace and increasing the number of variables names I need to declare, remember, type correctly, and so on. I also increase the number of variables the maintenance programmer has to untangle and understand. Second, every time I need to pass this information to a function I need to pass three different parameters:

```
my $connection = make_connection(
    $main_server_addr,
    $main_server_port,
    $main_server_username
);
```

Instead, I could write that same information using two anonymous hashes:

```
my $main_server = {
    addr    => "some.url",
    port    => 4000,
    username => "me"
};
```

```
my $backup_server = {
    addr    => "back.url",
    port    => "4444",
    username => "me"
};
```

This code is cleaner, defines only two variables, and lets me call functions easily:

```
my $connection = make_connection(
    $backup_server );
```

I could have used a named hash instead of the anonymous hash references:

```
%main_server = (
    addr    => "some.url",
    port    => 4000,
    username => "me"
);
```

and then I take the reference when I call the function:

```
$connection = make_connection(\%main_
server);
```

If I have more than one server, I might get better results handling the data if I make a hash of the server hashes, like this:

```
%servers = (
    main    => {
        addr    => "some.url",
        port    => 4000,
        username => "me"
    },
    backup1 => {
        addr    => "back.url",
        port    => 4444,
        username => "me"
    },
);
```