# Creating a hash from an array in Perl

hash

Prev

Next

When learning about hashes we have seen how we could create (or more specifically **initialize**) a hash from a list of key-value pairs. What if we have this data in an array?

Let's say we have an array called `@fruits` with the following 6 values: `apple, red, orange, orange, grape, purple`. We can assign this array to a hash and perl will automatically look at the values in the array as if they were key-value pairs. The odd elements (first, third, fifth) will become the keys and the even elements (second, fourth, sixth) will become the corresponding values.

**examples/hash_from_an_array.pl**

```perl
use strict;
use warnings;
use Data::Dumper qw(Dumper);

my @fruits = qw(apple red orange orange grape purple);

my %color_of = @fruits;


print Dumper \@fruits;
print Dumper \%color_of;
```

The output of this script is:

```
$VAR1 = [
          'apple',
          'red',
          'orange',
          'orange',
          'grape',
          'purple'
        ];
```

```
$VAR1 = {
          'orange' => 'orange',
          'apple' => 'red',
          'grape' => 'purple'
        };
```

The first `$VAR1` shows the content of the array. The second `$VAR1` show the content of the hash. If we run the script several times, the part of the array will always look the same, but the content of the hash not. The order of the pairs **in the display!** will change. That's because inside the hash there is no order and by default the `Dumper` function will display the key-value pairs in random order. The fruits will be always the keys and the colors always the values (I know the orange might be a bit confusing here, because both the fruit and its color are described by the same word.)

## Odd number of elements in hash assignment

What if someone removes one of the "orange" words from the array and tries to make the assignment that way?

```
my @fruits = qw(apple red orange grape purple);
my %color_of = @fruits;
```

This is going to be the output:

```
Odd number of elements in hash assignment at hash_from_an_array.txt line 7.
$VAR1 = [
          'apple',
          'red',
          'orange',
          'grape',
          'purple'
        ];
$VAR1 = {
          'orange' => 'grape',
          'apple' => 'red',
          'purple' => undef
        };
```

The first thing we can see is the warning Perl gives when it notices that the number of elements in the array which is being assigned to the hash cannot divided by 2. That the number of elements is odd.

Then, because this is just a warning (and it is only displayed if `use warnings` is in effect), the script will go on with the assignment. The odd elements ("apple", "orange" and "purple") will become keys and the even elements ("red", "grape") will become the values. Because the number of elements was odd, the last key ("purple") does not have a pair. It will get undef as its value.

In this case, not only have we received a warning, and an `undef` as one of the values, but the key-value pairs also got mixed up. ("purple", a color, became a key, and "grape", a fruit, became a value).

Which brings me to a discussion beyond the actual technique.

## Discussion

The assignment from an array to a hash will work technically (assuming there are even number of elements in the array), but most likely that array was only a temporary array. Normally an array hold values that have some common feature. For example "fruits", or "colors", but normally if you have a set of fruits and colors you don't keep them in an array. If you have two sets of things, both with its common feature (e.g. a bunch of fruits in one set and a bunch of colors in another set) then, if there is a mapping between the values, you can put them in a hash.

The case that we have, when an array holds both fruits and colors interleaved is a bit contrived. It can usually happen if the array is some kind of a temporary holder of the data.

Anyway, in order to make this clearer to the reader of your code, maybe it is better to use a different variable name. Even if it is longer. So for example instead of `my @fruits = qw(apple red orange grape purple);` we could have used `my @fruit_color_pairs = qw(apple red orange grape purple);`.