Manipulating Perl arrays: shift, unshift, push, pop

array shift unshift push pop

Prev Next

As well as allowing direct access to individual array elements, Perl also provides various other interesting ways to deal with arrays. In particular, there are functions that make it very easy and efficient to use a Perl array as a stack or as a queue.

pop

The `pop` function will remove and return the last element of an array.

In this first example you can see how, given an array of 3 elements, the `pop` function removes the last element (the one with the highest index) and returns it.

```
1. my @names = ('Foo', 'Bar', 'Baz');
   my $last_one = pop @names;

   print "$last_one\n";  # Baz
5. print "@names\n";     # Foo Bar
```

In the special case of the original array being empty, the `pop` function will return undef.

push

The `push` function can add one or more values to the end of an array. (Well, it can also add 0 values, but that's not very useful, is it?)

```
1. my @names = ('Foo', 'Bar');
   push @names, 'Moo';
   print "@names\n";     # Foo Bar Moo

5. my @others = ('Darth', 'Vader');
   push @names, @others;
   print "@names\n";     # Foo Bar Moo Darth Vader
```

In this example we originally had an array with two elements. Then we pushed a single scalar value to the end and our array got extended to a 3-element array.

In the second call to `push`, we pushed the content of the `@others` array to the end of the `@names` array, extending it to a 5-element array.

shift

If you imagine the array starting on the left hand side, the `shift` function will move the whole array one unit to the left. The first element will "fall off" the array and become the function's return value. (If the array was empty, **shift** will return undef.)

After the operation, the array will be one element shorter.

```
1. my @names = ('Foo', 'Bar', 'Moo');
   my $first = shift @names;
   print "$first\n";     # Foo
   print "@names\n";     # Bar Moo
```

This is quite similar to `pop`, but it works on the lower end of the array.

unshift

This is the opposite operation of `shift`. `unshift` will take one or more values (or even 0 if that's what you like) and place it at the beginning of the array, moving all the other elements to the right.

You can pass it a single scalar value, which will become the first element of the array. Or, as in the second example, you can pass a second array and then the elements of this second array (`@others` in our case) will be copied to the beginning of the main array (`@names` in our case) moving the other elements to higher indexes.

```
1. my @names = ('Foo', 'Bar');
   unshift @names, 'Moo';
   print "@names\n";     # Moo Foo Bar

5. my @others = ('Darth', 'Vader');
   unshift @names, @others;
   print "@names\n";     # Darth Vader Moo Foo Bar
```