

Manipulating Arrays

How to do things to an array in Perl

Page Resource

[CGI Resources](#) | [Perl Tutorials](#)

[Home/CGI/Perl/Arrays 2](#)

Now that you have seen the basics of arrays, you'll want to see how to change and manipulate an array or the elements of an array. The first thing we will look at is changing and adding elements.

Changing & Adding Elements

To change an array element, you can just access it with its list number and assign it a new value:

```
@browser = ("NS", "IE", "Opera");
$browser[2]="Mosaic";
```

This changes the value of the element with the list number two (remember, arrays start counting at zero- so the element with the list number two is actually the third element). So, we changed "Opera" to "Mosaic", thus the array now contains "NS", "IE", and "Mosaic".

Now, suppose we want to add a new element to the array. We can add a new element in the last position by just assigning the next position a value. If it doesn't exist, it is added on to the end:

```
@browser = ("NS", "IE", "Opera");
$browser[3]="Mosaic";
```

Now, the array has four elements: "NS", "IE", "Opera", and "Mosaic".

Splice Function

Using the splice function, you can delete or replace elements within the array. For instance, if you simply want to delete an element, you could write:

```
@browser = ("NS", "IE", "Opera");
splice(@browser, 1, 1);
```

You'll see three arguments inside the () of the splice function above. The first one is just the name of the array you want to splice. The second is the list number of the element where you wish to start the splice (starts counting at zero). The third is the number of elements you wish to splice. In this case, we just want to splice one element so we have 1. The code above deletes the element at list number 1, which is "IE" (NS is zero, IE is 1, Opera is 2). So now the array has only "NS" and "Opera", just two elements.

If you want to delete more than one element, change that third number to the number of

elements you wish to delete. Let's say we want to get rid of both "NS" and "IE". We could write:

```
@browser = ("NS", "IE", "Opera");
splice(@browser, 0, 2);
```

Now, it starts splicing at list number zero, and continues until it splices two elements in a row. Now all that will be left in the array is "Opera".

You can also use splice to replace elements. You just need to list your replacement elements after your other three arguments within the splice function. So, if we wanted to replace "IE" and "Opera" with "NeoPlanet" and "Mosaic", we would write it this way:

```
@browser = ("NS", "IE", "Opera");
splice(@browser, 1, 2, "NeoPlanet", "Mosaic");
```

Now the array contains the elements "NS", "NeoPlanet", and "Mosaic". As you can see, the splice function can come in handy if you need to make large deletions or replacements.

Unshift/Shift

If you want to simply add or delete an element from the left side of an array (element zero), you can use the unshift and shift functions. To add an element to the left side, you would use the unshift function and write something like this:

```
@browser = ("NS", "IE", "Opera");
unshift(@browser, "Mosaic");
```

As you can see, the first argument tells you which array to operate on, and the second lets you specify an element to be added to the array. So, "Mosaic" takes over position zero and the array now has the four elements "Mosaic", "NS", "IE", and "Opera".

To delete an element from the left side, you would use the shift function. All you have to do here is give the array name as an argument, and the element on the left side is deleted:

```
@browser = ("NS", "IE", "Opera");
shift(@browser);
```

Now, the array has only two elements: "IE" and "Opera".

You can keep the value you deleted from the array by assigning the shift function to a variable:

```
@browser = ("NS", "IE", "Opera");
$old_first_element= shift(@browser);
```

Now the array has only "IE" and "Opera", but you have the variable \$old_first_element with the value of "NS" so you can make use of it after taking it from the array.

Push/Pop

These two functions are just like unshift and shift, except they add or delete from the right side of an array (the last position). So, if you want to add an element to the end of an array, you would use the push function:

```
@browser = ("NS", "IE", "Opera");
push(@browser, "Mosaic");
```

Now, you have an array with "NS", "IE", "Opera", and Mosaic".

To delete from the right side, you would use the pop function:

```
@browser = ("NS", "IE", "Opera");
pop(@browser);
```

Now you have an array with just "NS" and "IE".

You can keep the value you deleted from the array by assigning the pop function to a variable:

```
@browser = ("NS", "IE", "Opera");
$last_element= pop(@browser);
```

Now the array has only "NS" and "IE", but you have the variable \$last_element with the value of "Opera" so you can make use of it after taking it from the array.

Chop

If you want to take the last character of each element in an array and "chop it off", or delete it, you can use the chop function. This comes in handy later when we are reading from a file, where we would need to chop the new line (\n) character from each line in a file. For now, let's just see how to use it. You would just write something like this:

```
@browser = ("NS4", "IE5", "Opera3");
chop(@browser);
```

This code would take the numbers off the end of each element, so we would be left with "NS", "IE" and "Opera".

Sort

If you want to sort the elements of an array, this can come in handy. You can sort in ascending or descending order with numbers or strings. Numbers will go by the size of the number, strings will go in alphabetical order.

```
@browser = ("NS", "IE", "Opera");
sort (ascend @browser);
```

```
sub ascend
{
    $a <=> $b;
}
```

The sub from above is a subroutine, much like what is called a "function" in other languages. We will explain that in more detail later. As you can see, the code above would sort in ascending order, so it would sort our array in alphabetical order. So, we have: "IE", "NS", and "Opera" as the new order. If you want it in reverse alphabetical order, you would use \$b <=> \$a in place of the \$a <=> \$b above. You could change the name of the subroutine to whatever you wish, just be sure you call the subroutine with the same name. This will make a bit more sense after we get to the section on subroutines and reading from files.

Reverse

You can reverse the order of the array elements with the reverse function. You would just write:

```
@browser = ("NS", "IE", "Opera");
reverse(@browser);
```

Now, the order changes to "Opera", "IE", and "NS".

Join

You can create a flat file database from your array with the join function. Again, this is more useful if you are reading and writing form files. For now, what you will want to know is that it allows you to use a delimiter (a character of your choice) to separate array elements. The function creates a variable for each element, joined by your delimiter. So, if you want to place a : between each element, you would write:

```
@browser = ("NS", "IE", "Opera");
join(":", @browser);
```

This would create something like this:

```
NS:IE:Opera
```

Split

The split function is very handy when dealing with strings. It allows you to create an array of elements by splitting a string every time a certain delimiter (a character of your choice) shows up within the string. Suppose we had the string:

```
NS:IE:Opera
```

Using the split function, we could create an array with the elements "NS", "IE", and "Opera" by splitting the string on the colon delimiter (:). We could write:

```
$browser_list="NS:IE:Opera";
@browser= split(/:/, $browser_list);
```

Notice in the split function that you place your delimiter between two forward slashes. You then place the string you want to split as the second argument, in this case the string was the value of the \$browser_list variable. Setting it equal to @browser creates the @browser array from the split.

Well, that should be enough to keep us all busy for a while. You can do a whole bunch of things with arrays, and we'll be using them extensively in later sections. So, have fun with what we have so far, we'll be moving on soon!

Well, that's it for now. Let's go on to: [Associative Arrays](#).

Partners

[CoolHomepages](#) | [Web Design Library](#) | [Website Content](#)

The tutorials and articles on these pages are © 1997-2005 by John Pollock and may not be reposted without written permission from the author, and may not be reprinted for profit.



[Previous](#)

By: [John Pollock](#)



[Next](#)

[Main Page](#) | [HTML](#) | [JavaScript](#) | [Graphics](#) | [DHTML/Style Sheets](#) | [ASP/PHP](#)
[PutWeb/FTP](#) | [CGI/Perl](#) | [Promotion](#) | [Java](#) | [Design Articles](#)
[Support Forums](#) | [Site Search](#) | [FAQs](#) | [Privacy](#) | [Contact](#)

Copyright © 1997-2005 [The Web Design Resource](#). All rights reserved.