

# CREATE REFERENCES

Each item of data in your script is stored in a specific location in the computer's memory. A reference, also called a real reference, is a scalar value that stores the type and location of an item stored in memory.

Using references allows you to work with complex data in Perl. For example, you can use references to pass multiple arrays to a subroutine. For information about using references with arrays, see page 166.

To create a reference, precede the value or variable you want to reference with a backslash (\), such

as \ \$x. Once a reference is created, Perl can access the item in the computer's memory and read the information stored at that location.

When you use the reference in a script, Perl will return the type of data that is being stored, such as a scalar or an array, and the location of the data in the computer's memory in hexadecimal notation.

To access the data the reference refers to, you must dereference the reference. For information about dereferencing a reference, see page 162.

## Extra

There are six types of references in Perl. You can create a reference to a scalar, array, hash, subroutine, reference or typeglob.

TO CREATE:	TYPE:
A reference to a scalar variable	\ \$scalarname
A reference to an array	\ @arrayname
A reference to a hash	\ %hashname
A reference to a subroutine	\ &subroutine
A reference to another reference	\ \$refname
A reference to a typeglob	\ *typeglob

A typeglob ties a variable name to a new variable name. This allows you to access variables using more than one name. For example, all the types of variables using a new name, such as \$moredata, @moredata or %moredata can access the information stored in variables using the original name, such as \$data, @data and %data. To create a typeglob to link variable names, prefix the new variable name and the existing variable name with an asterisk (\*).

### TYPE THIS:

```
$data = "Here's the information.";
@data = (1, 2, 3);
*moredata = *data;
print "$moredata\n";
print @moredata;
```

### RESULT:

Here's the information.
123

## CREATE A REFERENCE TO A VARIABLE

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl

$age = 36;

$ageRef = \ $age;

print $ageRef;
print "\n";
```

Get HelpExitWriteOutJustifyRead FileWhere isPrev PgNext PgCut TextUnCut TextCur PosTo Spell

[scripts]\$ script.pl

SCALAR(0x80e1648)

[scripts]\$

- 1

To create a variable that will store the reference, type the name of the new variable.
- 2

Type = \ followed by the name of the variable you want to create a reference to.
- 3

Position the cursor where you want to type the code that uses the reference and type the code.
- 4

Save and execute the script.
- Perl generates the type and location of the reference.

## CREATE A REFERENCE TO AN ARRAY

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl

@names = ("Ted Cains", "David Gregory", "Mary Corder");

$nameRef = \ @names;

print $nameRef;
print "\n";
```

Get HelpExitWriteOutJustifyRead FileWhere isPrev PgNext PgCut TextUnCut TextCur PosTo Spell

[scripts]\$ script.pl

ARRAY(0x80e1654)

[scripts]\$

- 1

To create a variable that will store the reference, type the name of the new variable.
- 2

Type = \ followed by the name of the array you want to create a reference to.
- 3

Position the cursor where you want to type the code that uses the reference and type the code.
- 4

Save and execute the script.
- Perl generates the type and location of the reference.

# DEREFERENCE A REFERENCE

After you have created a reference to an item of data stored in your computer's memory, you can access the data stored at the location. This is known as dereferencing a reference.

To dereference a reference you have previously created, prefix the name of the variable that stores the reference with a symbol for the type of information the reference refers to and a dollar sign (\$). For example, `$$ageRef` dereferences a reference to a scalar value, while `@$nameRef` dereferences a reference to an array.

References can also be dereferenced using blocks. To dereference a reference using a block, enclose the dollar sign (\$) and the name of the variable, or identifier, in braces after the symbol for the type of information the reference refers to, such as `${$ageRef}` or `@{ $nameRef }`.

Dereferencing a reference using a block also allows you to use an expression instead of the variable name as an identifier for a reference. For example, you can call a subroutine in the block to use an identifier generated by the subroutine.

Extra

When dereferencing a reference, you can assign a new value to the reference using the assignment operator (=). Changing the value of a reference changes the value stored in your computer's memory.

TYPE THIS:

```
$age = 33;
$name = " Tom Cains";

$ageRef = \$age;
$nameRef = \$name;

$$ageRef = 36;
$$nameRef = " Ted Cains";

print $age, $name;
```

RESULT:

36 Ted Cains

DEREFERENCE A REFERENCE

```
UW PICO(tm) 3.5      File: script.pl      Modified
#!/usr/bin/perl

$, = ", ";
$age = 36;
@names = ("Ted Cains", "David Gregory", "Mary Corder");

$ageRef = \$age;
$nameRef = \@names;

$
print "\n";
```

1 To access information you created a reference to, type the symbol for the type of information (example: \$ or @).

```
UW PICO(tm) 3.5      File: script.pl      Modified
#!/usr/bin/perl

$, = ", ";
$age = 36;
@names = ("Ted Cains", "David Gregory", "Mary Corder");

$ageRef = \$age;
$nameRef = \@names;

print $$ageRef;
print "\n";
```

2 Type \$ followed by the name of the variable that stores the reference.

3 Type the code that uses the result of dereferencing the reference.

```
UW PICO(tm) 3.5      File: script.pl      Modified
#!/usr/bin/perl

$, = ", ";
$age = 36;
@names = ("Ted Cains", "David Gregory", "Mary Corder");

$ageRef = \$age;
$nameRef = \@names;

print $$ageRef;
print "\n";
print @{$nameRef};
```

USING A BLOCK

4 To use a block to access information you created a reference to, type the symbol for the type of information and then type {\$.

5 Type the name of the variable that stores the reference and then type }.

6 Type the code that uses the result of dereferencing the reference.

```
[scripts]$ script.pl
36
Ted Cains, David Gregory, Mary Corder
[scripts]$
```

7 Save and execute the script.

Perl generates the result of dereferencing the references.

# USING SYMBOLIC REFERENCES

A symbolic reference stores the name of an item of data, such as a variable or array, and acts as a pointer to the item. This is different from a real reference, also called a hard reference, which stores the type and location of an item. Processing data using symbolic references is slower than processing data using real references, but it is useful for complex manipulations of references.

To create a symbolic reference, you create a variable that stores the name of the item you want the reference to point to. To later access or adjust the value of the item, you dereference the symbolic reference.

You can dereference a symbolic reference as you would dereference a real reference. You prefix the variable that stores the name of the reference with a symbol for the type of information the reference refers to. For more information about dereferencing, see page 162.

Symbolic references can be used only with variables that have a global scope, so they cannot be used with variables that use the `my` function. For information about the `my` function, see page 110.

## Extra

When using symbolic references, it is easy to make a mistake and use a symbolic reference when you intended to use a real reference. When you no longer need to use symbolic references and want to prevent them from being accidentally used, you can enter the `use strict 'refs';` declaration. If the script detects a symbolic reference in the block of code where you entered the `use strict 'refs';` declaration, an error message will be generated.

TYPE THIS:

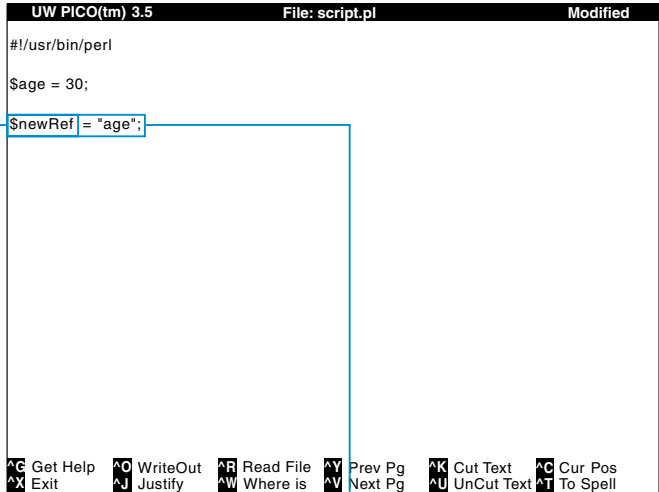
```
use strict 'refs';

$addressVar = "address";
$$addressVar = "9494 Riverdale Ave.";
print $address;
```

RESULT:

Can't use string ("address") as a SCALAR ref while "strict refs" in use at ./script.pl line 6.

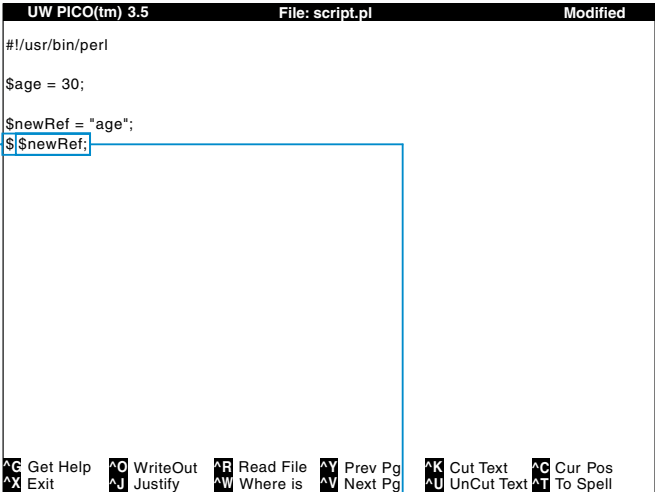
### USING SYMBOLIC REFERENCES



#### CREATE A SYMBOLIC REFERENCE

1 To create a variable that will store the symbolic reference, type the name of the new variable.

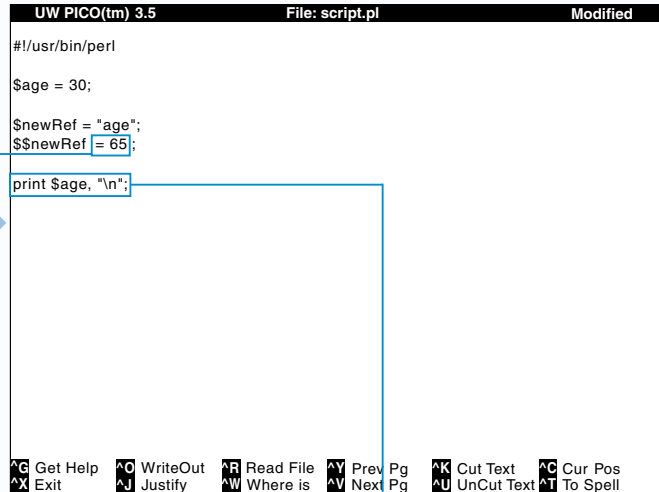
2 Type an equal sign (=) followed by the name of the item that you want the symbolic reference to refer to, enclosed in quotation marks.



#### DEREFERENCE A SYMBOLIC REFERENCE

3 To access information the symbolic reference points to, type the symbol for the type of information (example: \$ or @).

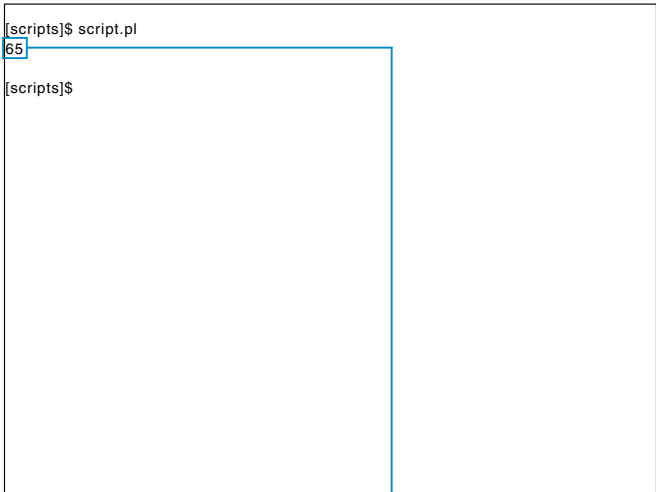
4 Type \$ followed by the name of the variable that stores the symbolic reference.



5 To adjust the value of the variable, type an equal sign (=) followed by the new value.

6 Position the cursor where you want to type the code that uses the symbolic reference and type the code.

7 Save and execute the script.



Perl generates the result of using the symbolic reference in the script.

# USING REFERENCES WITH ARRAYS

References are particularly useful for allowing you to pass multiple arrays to a subroutine. Normally, if you pass more than one array to a subroutine, all the data will be combined and stored in the `@_` array. In order to pass multiple arrays to a subroutine and preserve them as distinct arrays, you must pass the arrays using references. For more information about subroutines, see page 106.

To pass a reference for an array to a subroutine, you must precede the name of the array with a backslash (`\`) in the code that calls the subroutine.

In the code for the subroutine, you must extract the reference for each array from the `@_` array and assign the references to new variables. You must then dereference the references and assign the dereferenced elements to new arrays, which you can then use and work with in the subroutine. For information about dereferencing a reference, see page 162.

## Extra

The `ref` function allows you to check the reference type before dereferencing a reference. This is useful for avoiding errors when you pass an incorrect reference type to a subroutine. In this example, the `ref` function is used to confirm that the reference passed to the subroutine is an array before the result is printed. Since the second reference passed to the subroutine is not an array, Perl does not print the result.

TYPE THIS:

```
CheckRef (\("A" .. "E"), \10);

sub CheckRef
{
    my($ref_letters, $ref_number) = @_;

    print "The first array is @{$ref_letters}.\n" if (ref($ref_letters) eq "ARRAY");
    print "The second array is @{$ref_number}.\n" if (ref($ref_number) eq "ARRAY");
}
```

RESULT:

The first array is A B C D E.

### USING REFERENCES WITH ARRAYS

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl

@names = ("Ted", "Lindsay", "Mary");
@ages = (29, 37, 32);

printList(\@names, \@ages);

sub printList ( )
{
}
```

- 1
- Type the code that creates multiple arrays and assigns their values.
- 2
- Create and call the subroutine you want to pass the arrays to. To create and call a subroutine, see page 106.

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl

@names = ("Ted", "Lindsay", "Mary");
@ages = (29, 37, 32);

printList(\@names, \@ages);

sub printList ( )
{
    ($namesRef, $agesRef) = @_;
}
```

- 3
- In the code that calls the subroutine, position the cursor over the closing parenthesis and type a reference for each array you want to pass to the subroutine, separated by a comma.
- 4
- In the code for the subroutine, type `( ) = @_;`.
- 5
- Position the cursor over the closing parenthesis and type the name of each variable you want to store a reference passed to the subroutine, separated by a comma.

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl

@names = ("Ted", "Lindsay", "Mary");
@ages = (29, 37, 32);

printList(\@names, \@ages);

sub printList ( )
{
    ($namesRef, $agesRef) = @_;

    @employees = @{$namesRef};
    @yearsOld = @{$agesRef};

    for ($x = 0; $x <= 3; $x++)
    {
        print "@employees[$x]\t@yearsOld[$x]\n";
    }
}
```

- 6
- Type the code that dereferences the references and assigns the dereferenced elements to new arrays.
- 7
- Type the code that will process the elements in the arrays.

[scripts]\$ script.pl

```
Ted 29
Lindsay 37
Mary 32

[scripts]$
```

- 8
- Save and execute the script.
- 9
- Perl generates the results of using references to pass arrays to a subroutine.

# CREATE AN ARRAY OF HASHES

References are useful for creating complex data structures in Perl, such as an array of hashes.

An array of hashes allows you to use a hash as an element in an array. This is useful if you want to create an array of records. For example, each hash in an array could contain information for one customer. In order to create an array of hashes, you must assign each hash to the array using references. For more information about using references with arrays, see page 166.

To assign a reference for a hash to an array, you must precede the name of the hash with a backslash (\) in the code that stores the information in the array.

Once the array of hashes is created, you can access an individual item from the hash by dereferencing the references and specifying the index of the hash element you want to access. You then use the arrow operator (->) to specify the key of the element you want to access. For information about dereferencing a reference, see page 162.

## Extra

You can use the arrow operator (->) instead of the dollar sign (\$) to dereference a reference to an array, hash or subroutine. This is useful when you have several references and you want to clarify the relationship between an array, hash or subroutine and the index, key or list of arguments you want to dereference.

TYPE THIS:

```
@phoneList = qw(555-1212 555-6161 555-9090);
sub PrintLast(@)
{
    print @_[ $# ], "\n";
}
$arrayRef = \@phoneList;
$subRef = \&PrintLast;

print $arrayRef->[1], "\n"; $subRef->(@phoneList);
```

RESULT:

```
555-6161
555-9090
```

## CREATE AN ARRAY OF HASHES

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl

%page1 = ("Title" => "Home page",
          "Filename" => "index.html",
          "Size" => 23673);

%page2 = ("Title" => "Sales information",
          "Filename" => "sales.html",
          "Size" => 23675);

%page3 = ("Title" => "Contact",
          "Filename" => "address.html",
          "Size" => 23677);

@webPages = (\%page1, \%page2, \%page3);
```

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

- 1
- Type the code that creates multiple hashes and assigns their keys and values.
- 2
- Type the name of the array to which you want to assign the hashes followed by = ( );.
- 3
- Position the cursor over the closing parenthesis and type a reference for each hash you want to assign to the array, separated by a comma.

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl

%page1 = ("Title" => "Home page",
          "Filename" => "index.html",
          "Size" => 23673);

%page2 = ("Title" => "Sales information",
          "Filename" => "sales.html",
          "Size" => 23675);

%page3 = ("Title" => "Contact",
          "Filename" => "address.html",
          "Size" => 23677);

@webPages = (\%page1, \%page2, \%page3);

%{$webPages[1]}->{}
```

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

- 4
- Position the cursor where you want to access an element in the array and type %{ }->{ }.
- 5
- Position the cursor over the first closing brace and type the name of the array that contains the element you want to access followed by the index of the element.

UW PICO(tm) 3.5File: script.plModified

```
#!/usr/bin/perl

%page1 = ("Title" => "Home page",
          "Filename" => "index.html",
          "Size" => 23673);

%page2 = ("Title" => "Sales information",
          "Filename" => "sales.html",
          "Size" => 23675);

%page3 = ("Title" => "Contact",
          "Filename" => "address.html",
          "Size" => 23677);

@webPages = (\%page1, \%page2, \%page3);

print "The filename of the second Web page is:\n";
print %{$webPages[1]}->{"Filename"};
```

Get HelpWriteOutRead FilePrev PgCut TextCur PosExitJustifyWhere isNext PgUnCut TextTo Spell

- 6
- Position the cursor over the last closing brace and type the key of the element you want to access enclosed in quotation marks.
- 7
- Position the cursor where you want to type the code that uses the element you specified and type the code.

[scripts]\$ script.pl

```
The filename of the second Web page is:
sales.html
[scripts]$
```

- 8
- Save and execute the script.
- Perl generates the result of creating an array of hashes.