

Idiomatic Perl

Writing Better Perl

Dave Cross

dave@dave.org.uk

Bibliotech

<http://www.dave.org.uk/talks/idiomatic/idiomatic.pdf>

Open Source Convention / San Diego / 22 July 2002

Introduction

Idiomatic Perl / use strict and -w

Be careful out there

- `use strict` and `-w` can catch many tricky bugs
- A very good habit to get into
- Programming with a safety net
-

Idiomatic Perl / use strict and -w

use strict 'refs'

- Prevents symbolic references
- aka "using a variable as another variable's name"
-

```
$what = 'slayer';  
$$what = 'Buffy';  
# sets $slayer to 'Buffy'
```

- What if 'slayer' came from user input?

Idiomatic Perl / use strict and -w

use strict 'refs' (cont)

- Better to use a hash

```
$what = 'slayer';  
$people{$what} = 'Buffy';
```
- Self contained namespace
- Less chance of clashes
- More information (e.g. all keys)

Idiomatic Perl / use strict and -w

Some typical warnings

Idiomatic Perl / use strict and -w

More info

- `use strict`
- `warnings`

Perl Variable Types

Idiomatic

Idiomatic Perl / Perl Variable Types

When to use local or my

- Easy answer - always use `my`
- Slightly more complex answer - always use `my` (except when it doesn't work)
- Full answer (by Mark-Jason Dominus) -
<http://perl.plover.com/FAQs/Namespaces.html> <http://perl.plover.com/local.html>

Perl References

Idiomatic Perl / Perl References

References

Idiomatic Perl / Perl References

Creating References (cont)

- ```
@arr = (1, 2, 3, 4);
$aref1 = \@arr;
$aref2 = [@arr];
print "$aref1 $aref2";
```
- Output:  

```
ARRAY(0x20026800)
ARRAY(0x2002bc00)
```
- Second method creates a **copy** of the array

## Idiomatic Perl / Perl References

### Using References





## **Idiomatic Perl / Perl References**

### **Why Use References? (cont)**

- Another attempt



# Idiomatic Perl / Perl References

## More Complex Data Structures

- Suppose you have a data file like this-1 0 0 -1 509.495 11111r9<e a da 064.49513.567 hal

## Idiomatic Perl / Perl References

### Using More Complex Data Structures

- Using an array of hashes

```
foreach (@records) {
 print "$_->{f_name} $_->{s_name} ";
 print "is a $_->{job}\n";
}
```

## Idiomatic Perl / Perl References

### Even More Complex Data Structures

- Many more possibilities
- Hash of hashes
- Hash of arrays
-



# Idiomatic Perl / Finding, Installing and Using Modules

## Modules

- A module is a reusable 'chunk' of code
- Perl comes with over 100 modules (see `perldoc perlmodules` for list)  
Perl has a repository of freely-available modules - the Comprehensive Perl Archive Network (CPAN)  
<http://www.cpan.org>  
<http://search.cpan.org>

# Idiomatic Perl / Finding, Installing and Using Modules

## Installing Modules (the hard way)

-

# Idiomatic Perl / Finding, Installing and Using Modules

## Installing Modules (the hard way) (cont)

- **Note:** May need root permissions for make install
- You can have your own personal module library

```
perl Makefile.PL PREFIX=~/.perl
```

(need to adjust @INC)



# **Idiomatic Perl / Finding, Installing and Using Modules**

## **Using Functional Modules**



# Idiomatic Perl / Finding, Installing and Using Modules

## Useful CPAN Modules

- Template





# Idiomatic

# Idiomatic

# Idiomatic Perl / Writing Reusable Code

## Using @EXPORT and @EXPORT\_OK

- How does import k859 which subroutines to export?
- Exports are defined in @EXPORT or @EXPORT\_OK
- Automatic exports are defined in @EXPORT
- Optional exports are defined in @EXPORT\_OK



# Idiomatic Perl / Writing Reusable Code

## Writing Modules The Easy Way

- Use `h2xs` to create a skeleton module
- `h2xs -A -X -n MyModule`
- Creates six files
- `MANIFEST`, `Makefile.PL`, `MyModule.pm`, `test.pl`, `README` and `CHANGES`



# Idiomatic Perl Writing Reusable Code

## More Information

- `perldoc perlmod`
- `perldoc perlboot`
- `perldoc perltoot`
- `perldoc perlobj`
-

# Idiomatic Perl / Sorting

## Basic Sorting

-



# Idiomatic Perl /

## **Idiomatic Perl / Sorting**

### **More Complex Sorts [2]**

# Idiomatic Perl / Sorting

## Putting It All Together

- Can rewrite this as
- 

@names =



# Idiomatic Perl / Cartoon Swearing

## Special Variables

- Perl has a number of special built-in variables that we can use
- Named with punctuation characters (`$_`) or control characters (`$_^w`)
- Always in main package
- Can't create with `my` - always use `local`
- Only alter them localised in a block
-

# Idiomatic



# Idiomatic Perl / Cartoon Swearing

## Current Record Number - \$.

- No need to keep a count of the current record number
- 

```
while (<FILE>) {
 print "$.: $_";
}
```

- **Note:** contains current record number from most recently read filehandle
- **Note:**

# Idiomatic Perl / Cartoon Swearing

## More Output Control Variables

- \$, is the output field separator
  - Controls what is output between the arguments to `print`
- 

```
@nums = 1 .. 3;
print @nums; # 123
$, = ' - ';
print @nums; # 1 - 2 - 3
```



## Idiomatic Perl / Cartoon Swearing

### Process Variables

- \$0 contains the name of the current program name
- \$\$ contains the process ID
- \$<, \$( contain the real user and group IDs
- \$>, \$) contain the effective user and group IDs

## Idiomatic Perl / Cartoon Swearing

Useful 19.2.7 The following variables contain the environment information







## Idiomatic Perl / Context

### Contextual Differences

- There is no way to work out what an operator or function will do in a given context
- 

```
@a = reverse(1, 2, 3) # (3, 2, 1)
```

- 

```
$s = reverse('123') # '321'
```

- 

```
print reverse('123') # '123'
```

## Idiomatic Perl / Context

### Knowing Your Context

- You can find out what context your function has been called in by using the (badly named) `wantarray`



# Idiomatic



# Idiomatic

# Idiomatic Perl / foreach vs grep vs map

## map & grep Caveats (2)

- A better solution
- 

```
sub at_least_one_match {
 my $pattern = shift;
 foreach (@_) {
 return 1 if /$pattern/
 }
 return; # No match
}

if (at_least_one_match('pattern', @long)) {
 ...
}
```

## Idiomatic Perl / foreach vs grep vs map

### More Information

- `perldoc perlsyn (foreach)`
- `perldoc -f map`
- `perldoc -f grep`

## Boolean Expressions as Conditionals

## Idiomatic Perl / Boolean Expressions as

- In Perl Boolean expressions are
- Perl only does as much work as necessary  
(Laziness is a virtue!)

## Idiomatic Perl / Boolean Expressions (Conditionals 2)

- `EXPR1 or EXPR2`
  - Only need to evaluate `EXPR2` if `EXPR1` is false
- ~~`EXPR1 and EXPR2`~~
- Only need to evaluate `EXPR2` if `EXPR1` is true



# Idiomatic Perl / Boolean Expressions as Conditionals

## Some Other Examples

-





## **Idiomatic Perl / Statement Modifiers**

### **More Information**

- `perldoc perlsyn`

## **Assignment Operators**

# Idiomatic Perl / Assignment Operators

## Assignment Operators

- Perl has a number of assignment operators that it shares with other languages
- +=, -=, \*=, /=
- But Perl extends the list greatly
- %=, .=, x=, || =
-

# IdiomaticOperatorsc



## **Idiomatic Perl / Quoting**

### **Overusing Backslashes**

- Have you ever done this?





