

# CS306: Introduction to Perl

## Section #8: Elements of Style

U. of Alabama at Birmingham  
Dept. of Computer & Information Sciences

Slide 1

# Section 8: Elements of Style

strict and warnings  
Error Handling  
Debugging  
Documentation

Slide 2

## use strict

- Perl provides some ways to help you write better code – use strict is one of these.
- This is called the use strict *pragma*
- Using the strict pragma will prevent you from making lots of common mistakes

Slide 3

## How to use strict;

- Just add the use strict; statement at the top of your program

```
#!/usr/bin/perl  
use strict;  
...
```

Slide 4

## What Does use strict Prevent?

- use strict; is strict about three things
  - Thing #1: “vars” - This will complain about variables that don't meet one of these criteria
    - A Perl special variable
    - Declared with “my”
    - Some other things which we haven't talked about yet (variables from other packages)
  - So note that “vars” will prevent you from using global variables. This is A Good Thing(tm)

Slide 5

## Example of variable strictness

- `#!/usr/bin/perl`  
`use strict;`  
`$foo = 1; #oops, global variable!`  
  
`use strict` complains at runtime:  
“Global symbol “\$foo” requires explicit package name.”

Slide 6

## Example of variable strictness

- `#!/usr/bin/perl`  
`use strict;`  
`my $foo = 1; # that's better`

Slide 7

## Bareword Strictness

- Thing #2: “subs” - Prevents you from using barewords where you mean to use a string  
  
`use strict;`  
`$str = Hello; # Oops! Bareword string`  
  
“Bareword “hello” not allowed while “strict subs” in use”

Slide 8

## Bareword Strictness Continued

- use strict;  
\$str = 'hello'; #better
- Also applies to function calls  
use strict;  
my \$result = foo; # use strict complains here too  
sub foo { ... }

Slide 9

## Bareword Strictness Continued

- use strict;  
\$str = 'hello'; #better
- Also applies to function calls  
use strict;  
my \$result = foo(); # now we're ok  
sub foo { ... }

Slide 10

## Bareword Strictness Continued

- Initializing arrays and hashes

```
use strict;  
@arr = (foo, bar);      # BAD  
@arr = qw/foo bar/;    # OK  
%hash = (foo => bar);  # BAD  
%hash = (foo => "bar"); # OK – values req. “ “
```

Slide 11

## use strict Summary

- These things might seem like common sense, but use strict; will also protect you from much more subtle errors down the road.
- Use it all the time, it will save you hours of debugging

Slide 12

## use warnings;

- Similar to use strict, but will not stop execution of the program. Just throws warnings to STDERR.
- Will warn about all sorts of things...
  - “Use of uninitialized value in addition (+)”
  - “Name "main::people" used only once: possible typo”
  - Hundreds more

Slide 13

## Basic Debugging with Data::Dumper

- Data::Dumper is a module that will “pretty print” a data structure

```
- use Data::Dumper;  
my %hash = (apple => 'red', banana => 'yellow');  
print Dumper(\%hash);
```

```
$VAR1 = {  
  'banana' => 'yellow',  
  'apple' => 'red'  
};
```

Slide 14

## Data::Dumper Continued

- This will be invaluable as we build more complex data structures (hashes of hashes, multidimensional arrays, arrays of hashes of arrays of hashes of hashes of... ;-)

Slide 15

## Data::Dumper Continued

- use Data::Dumper;  
my %hash = (apple => 'red', banana => 'yellow');  
print Dumper(\%hash);
- What's the \ for? Dumper works better if you do that for arrays and hashes. It's called a reference, and we'll talk about it in a few lectures. For now, just remember to use it with arrays and hashes.

Slide 16

## Data::Dumper Continued

- use Data::Dumper;  
my %hash = (apple => 'red', banana => 'yellow');  
print Dumper(\%hash);
- Another interesting thing is happening here.  
Where did the Dumper() function come from?  
When we say “use Data::Dumper”, we get access  
to Dumper() for free. To be discussed later.

Slide 17

## Quick Debug Mode

- #!/usr/bin/perl  
use Data::Dumper;  
\$debug = 0; #flip this if you want debugging  
...  
print Dumper(\%foo) if \$debug;  
...  
print “I'm entering sub baz now” if \$debug;  
...

Slide 18

## Error Handling in Perl - die()

- You can use die() for the most basic, “I can't  
continue any further, woe is me” error handling
  - open FILE, “<\$filename” or die “Can't find the file!”;
  - chdir “/some/directory” or die “Directory not found.”;
  - die “Not enough arguments” unless @ARGV >= 2;
- The program immediately exits with your  
message.

Slide 19

## warn()

- Identical to die, except it doesn't exit the  
program.

This space for rent. (Just seeing if you're awake)

Slide 20

## Smarter Error Handling – eval blocks

- Putting code in an eval block traps normally fatal errors
  - Similar to try block in C++ or Java
- Immediately after the eval block, \$@ is guaranteed to be null if the block was successful or hold the error message if there was an error.

Slide 21

## eval {...}; Examples

- eval { \$result = \$num1 + \$num2; };
  - Division By Zero no longer a fatal error
- eval {  
    \$result = \$num1 + \$num2;  
};  
if (\$@) {  
    print "Error: \$@, I'm continuing now.";  
}

Slide 22

## eval {...}; Examples Continued

- for \$file (@files) {  
    eval {  
        open IN, "<\$file" or die;  
        # do something to file here  
        close IN;  
    };  
    if (\$@) { print "\$file skipped. (\$@)"; }  
}

Slide 23

## eval {...}; Summary

- As with most things we're learning, eval { ... }; is a powerful structure and we're just beginning to see its usefulness
- For example, \$@ can hold an error object, not just a message. But that's for another class.

Slide 24

## Using Perl's Documentation

- Perl provides a lot of online documentation
- Use the 'perldoc' command to get to it
- # perldoc *module-name*
- # perldoc -f *function-name*
- # perldoc [perlfaq | perlstyle | perlop | perldata | perlsync | perlrequick | perlvar | etc... ]
  - many more, see "perldoc perltoc" for full list

Slide 25

## Style Summary

- use strict and warnings – they buy you lots of style all at once
- Data::Dumper will become your best debugging friend
- Use error handling techniques to allow your program to act more gracefully when the unexpected happens
- The perldoc is a treasure trove, explore it

Slide 26