ITE315 Module 3 Part C - Programming in PERL: Larger Programs: Packages, Namespaces, and CPAN

Athens State University

Contents

1 Namespaces

Namespaces and Packages

- Things in Perl are organized into **namespaces**
- **Package**: A collection of code into a single namespace
- All global variables and functions defined within a package have scope only in that namespace
  - These can only be accessed using fully qualified names
- The scope of a package continues until the next `package` declaration or the end of the file

Packages

```
1 package MyCode;
2
3 our @boxes;
4 sub addBox { ... }
```

- You must refer to `@MyCode::boxes` and `@MyCode::addBox` using fully qualified names

BTW, the ellipsis operator (the three dots thing) is a valid special operator in Perl, says "unimplemented" and any code that tries to execute `@MyCode::addBox` will throw an exception.

Switching Between Namspaces

```
#!/usr/bin/perl
2 use strict;
use warnings;
4 use 5.010;
sub hi { return "main";}
6
package Foo;
```

```
8
sub hi { return "Foo";}
10
say main::hi();    # main
12 say Foo:hi();      # Foo
say hi();          # Foo
14
package main;
16
say main::hi();    # main
18 say Foo:hi();      # Foo
say hi();          # main
```

The default namespace in Perl is defined by the `main` package. This means everything you don't include in your own packages gets stored in that namespace.

You can switch between packages in a file by using the `package` keyword. This is why you see the behavior in the example code.

BTW, don't do this... it's bad form from a design standpoint. Here's Dr. Lewis's Rule of Perl Design Sanity: one package, one file, and then use the darn things. It's cleaner from a maintenance standpoint. Of course your workplace may have different rules...

Three Methods Provided To A Package

- Each package needs to define a version
  - This must be in the form: `v<INT>.<INT>.<INT>`
- Each package will have default methods: `VERSION()`, `import()`, and `unimport()`

Perl has strict rules for version numbers. They must have the leading 'v' character and at least three integer components separated by periods. One defines a version in the a package in the following manner:

```
1 package MyCode
{
3   our $VERSION = 1.21.0;
}
```

The `VERSION()` method returns the version number to the caller. If you provide a version number as an argument to this function, it will throw an exception unless the version number of package is equal to or greater than the argument:

```
# Need at least v2.1
2 Some::Plugin->VERSION( 2.1 );
die "Your plugin $version is too old" unless $version > 2;
```

The `import()` and `unimport()` are system things that you can redefine if you need to some sort of special processing when a package is loaded or unloaded.

2 CPAN

CPAN

- **CPAN**: Comprehensive Perl Archive Network

  - Repository for a LARGE set of Perl modules
  - First place to look for solutions to a problem

- CPAN provides a repository manager that you can use to download and install Perl modules

A few notes about CPAN:

- CPAN will configure itself when first run, unless you really know what you're doing, select the default options for putting downloaded code into your local folders
- Using local folders will require you to make modifications to your Bash configuration. The configuration utility will list a set of environment variables that to be placed into your `.bash_profile`.

There are two ways to run CPAN:

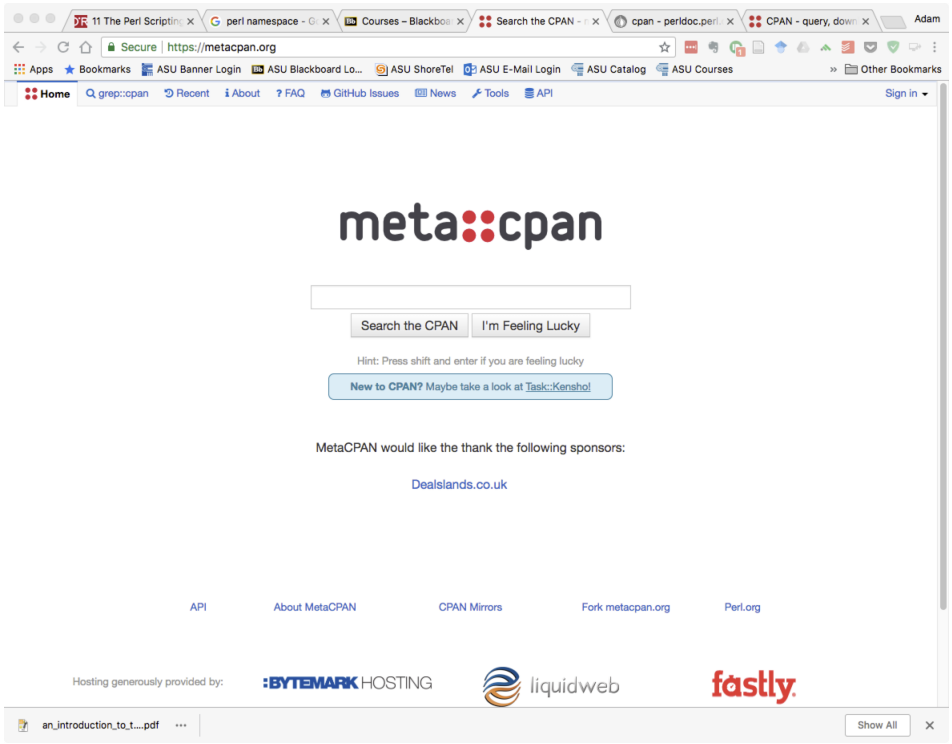
```
perl -MCPAN -e 'install Timestamp::Simple'
cpan
```

If you know what package you want to install, then you can execute CPAN functions directly from the command-line. More often you will use the CPAN utility to download, unpack, and install modules.

Caveat: Windows

- The ActivePerl distribution uses its own package manager, PPM
- The other popular Perl distribution for Windows, Strawberry Perl, can use CPAN with some configuration chnages

Finding Stuff on CPAN: metacpan.org



- There is a LOT of stuff on CPAN
- Traditionally, one used `http://search.cpan.org` to find things
- CPAN is in the process of switching to the `MetaCPAN` search engine
  - This is the preferred method of finding stuff going forward

Case Study: Editing Timestamps

```
coas-alewis-mbp:~ alewis$ cpan
cpan shell -- CPAN exploration and modules installation (v2.00)
Enter 'h' for help.

cpan[1]> m Timestamp::Simple
Reading '/Users/alewis/.cpan/Metadata'
  Database was generated on Mon, 18 Jun 2018 12:17:02 GMT
Module id = Timestamp::Simple
    CPAN_USERID  SHOOP (Steve Shoopak <shoop@cpan.org>)
    CPAN_VERSION 1.01
    CPAN_FILE    S/SH/SHOOP/Timestamp-Simple-1.01.tar.gz
    UPLOAD_DATE  2007-02-08
    INST_FILE    (not installed)

cpan[2]> install Timestamp::Simple
Running install for module 'Timestamp::Simple'
Running make for S/SH/SHOOP/Timestamp-Simple-1.01.tar.gz
```

The textbook explains how to manually download a module from CPAN. Don't do that... use the tools we have to manage these things.

We want to work with timestamps: time and date strings in the format "`YYYMMDDHHMMSS`". There is a module in CPAN called `Timestamp::Simple` that does basic parsing and conversion of strings in this format.

The transcript in the slide shows the steps for searching for this module in the CPAN package manager and then installing it to your designated library location (local to your home folder in our case).

Case Study: Editing Timestamps

```
#!/usr/bin/perl
use Timestamp::Simple qw($stamp);
# Save the timestamp into a working variable
$ts = $stamp, "\n";
# Strip the year
$ts =~ s/....(.*)/\1/;
# Create a temp file name
$fn = "tmpfile." . $ts;
open $OUTFILE, '>', "$fn";
print $OUTFILE "Hi there. \n";
close $OUTFILE
```

3 Key Points

Key Points

- What are packages and namespaces?
- What is CPAN?

- Basic use of CPAN to add to the core language