

CS306: Introduction to Perl

Section #12: Fun Stuff I

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

Slide 1

Section #12: Fun Stuff I

Intro
CPAN
Networking
CGI
Database

Slide 2

Fun Stuff Introduction

The intent of this lecture is a rapid-fire introduction to several interesting application domains of perl, in the hopes that you will be inspired with an interesting final project.

It is not meant to be in-depth or all-inclusive. You should read the textbook and/or perldoc for further information about the various topics presented today.

Slide 3

How is Perl Organized?

- Core functions
 - This is what you get for “free” just by saying 'perl' on the command line or #!/usr/bin/perl at the top of your program
- Standard modules
 - Modules that are so useful they ship with Perl. You use them by saying things like 'use Data::Dumper' in your program
- Optional modules that don't ship with Perl

Slide 4

What is the CPAN?

- The Comprehensive Perl Archive Network
- <http://www.cpan.org/>
- A large collection of Perl software and documentation
- The best place to get perl modules that are not part of the core distribution

Slide 5

What's available on the CPAN?

- Everything
- If you can think of a piece of functionality that would be useful, there's a 99.9% chance that it is already there
- For example, there are:
 - 85 modules for working with Email
 - 220 modules for working with Dates and Calendars
 - Hundreds for working with the Internet and WWW

Slide 6

How do I use the CPAN?

- Two ways
 - Manual method – browse to CPAN, find the module you want, download the .gz file and install it
 - perl Makefile.PL, make, make test, make install
 - many require a C compiler
 - Using a package manager
 - CPAN.pm
 - PPM for ActiveState Perl (Windows)

Slide 7

Using CPAN.pm

- CPAN.pm is a standard module (comes with Perl) that is used to simplify the download and install process from CPAN
- Generally used with command-line perl, like this:
 - perl -MCPAN -e 'shell'
 - This says I want to load the CPAN module (-M) and call the 'shell' function (-e).
 - 'shell' is a function in the CPAN module that provides a command-line interface to CPAN

Slide 8

CPAN.pm Example

- `perl -MCPAN -e shell`

```
# perl -MCPAN -e shell
cpan shell -- CPAN exploration and modules installation
(v1.7601)
cpan> install Email::Simple
Running install for module Email::Simple
Running make for C/CW/CWEST/Email-Simple-1.92.tar.gz
Fetching with LWP:
ftp://archive.progeny.com/CPAN/authors/id/C/CW/CWEST
(Continued....)
```

Slide 9

CPAN.pm Example

```
CPAN.pm: Going to build C/CW/CWEST/Email-
Simple-1.92.tar.gz
Checking if your kit is complete...
Looks good
[SNIP]
Running make test
[SNIP]
All tests successful.
Running make install
[SNIP]
/usr/bin/make install -- OK
cpan> quit
```

Slide 10

Installing Modules Without root

- If you don't have root access to the machine, CPAN.pm will not be able to install modules into the system-wide perl module directories (this is something like `/usr/lib/perl5/5.8.3/...`)
- You can still install modules to your home directory for your own use

Slide 11

Installing Modules Without root

- The manual way

```
# perl Makefile.PL PREFIX=/home/fran/perl
LIB=/home/fran/perl/lib
and then in your program...
use lib '/home/fran/perl/lib';
use WhateverModuleYouInstalled;
or set the PERL5LIB environment variable...
# PERL5LIB=/home/fran/perl/lib
```

Slide 12

Installing Modules Without root

- The CPAN.pm way

```
cpan> o conf makepl_args 'PREFIX=/home/fran/perl
LIB=/home/fran/perl/lib'
makepl_args PREFIX=/home/fran/perl
LIB=/home/fran/perl/lib

cpan> install Email::Simple
```

Slide 13

What if I Don't Know What I Need?

- Then search the CPAN
 - <http://search.cpan.org/>
 - <http://cpan.uwinnipeg.ca/htdocs/faqs/cpan-search.html>
- Often there are too many choices! Which one is best? Which one is most popular? Sometimes Google searches help. Sometimes you just learn over time. TIMTOWTDI is a double-edged sword.

Slide 14

Notes for Perl on Windows

- CPAN is designed to support Perl installations that are in Unix-like environments
- ActiveState Perl has the PPM, Programmer's Package Manager

```
C:> PPM install Email-Simple
C:> PPM install --location=http://some.other.repository/
Module-Name
```

- Note that :: changes to – on Windows

Slide 15

Notes for Perl on Windows

- However, PPM doesn't have all of the modules of CPAN. You can install CPAN modules onto Windows, but it's not the simplest procedure
- Useful URLs
 - <http://aspn.activestate.com/ASPN/docs/ActivePerl/faq/ActivePerl-faq2.html>
 - <http://www.perlmonth.com/modules.php?name=News&file=article&sid=59>

Slide 16

Networking

- You all seem very interested in this. I'm not sure why (it's all about the web services in the 21st century!) but you ask, and so you shall receive!
- Perl provides all of the same low-level functions and functionality that C does for working with sockets
- Perl also provides IO::Socket modules to make the process a little more user-friendly

Slide 17

Creating a TCP client

- ```
use IO::Socket;
$socket = IO::Socket::INET->new(
 PeerAddr => $remote_host,
 PeerPort => $remote_port,
 Type => SOCK_STREAM,);

print $socket "E.T. is phoning home.\n";
$answer = <$socket>;
print $answer;
close $socket;
```
- ```
$client =
    IO::Socket::INET->new('www.cis.uab.edu:80')
    or die();
```

Slide 18

Creating a TCP Server

- ```
use IO::Socket;
$server = IO::Socket::INET->new(
 LocalPort => $server_port,
 Type => SOCK_STREAM,
 Reuse => 1,
 Listen => 10,) or die();

while ($client = $server->accept()) {
 $request = <$client>;
 print $client "Hello, E.T. Enjoy Earth\n";
}
close $server;
```

Slide 19

## A Forking Server

```
use IO::Socket;
$server = IO::Socket::INET->new(
 LocalPort => 1234, Type => SOCK_STREAM,
 Reuse => 1, Listen => 10)
 or die;
while ($client = $server->accept()) {
 if ($kidpid = fork) {
 close $client;
 next;
 }
 defined $kidpid or die("Can't fork");
 close $server;
 $request = <$client>;
 print $client "Got it.\n";
 exit;
}
close $server;
```

Slide 20

## Learn More About Networking

- Read the Sockets section of Chapter 16 in your textbook (page 437)
- I recommend using the higher-level IO::Socket module instead of the Socket module
- There's a chapter of recipes in the Perl Cookbook
- There are other books written on the topic, as well

Slide 21

## Dynamic Web Apps With CGI.pm

- CGI is the Common Gateway Interface
- The simplest way to provide interactive applications on the web
- Stateless - works as http requests. Request received from browser, response sent to browser, connection closed, program terminated.
- Perl provides CGI.pm to create CGI scripts

Slide 22

## The CGI Model

1. The web server gets a request for a Perl CGI script.
2. The script is executed, meaning that the Perl interpreter is launched, and the web server hands over the request and data based on the CGI protocol.
3. STDOUT is redirected back through the CGI gateway. The script produces its output and exits. The Perl interpreter terminates.

Slide 23

## CGI.pm - Simple Example

- ```
use CGI qw/:standard/;

print header, start_html('Custom Doc Quoter'),
  h1('Doc Quoter'),
  start_form,
  "Enter a number: ", textfield('num'),
  P,
  submit, end_form, hr;

if (param()) {
  print param(num) . " gigawatts!!! ",
    "The only place to get " . param(num) .
    " gigawatts is a bolt of lightning!!";
}
```

Slide 24

CGI.pm Features

- CGI.pm provides a lot of helper functions that produce HTML as output. Many take arguments... `header()`, `h1('heading')`, `p`, `start_form`, `start_html('title')`, etc.... many correspond to the equivalent HTML tag.
- CGI.pm gathers up all the CGI input data (passed by either the GET or POST method) and exposes it through the `param()` function for you to get at it.

Slide 25

A General CGI Framework

- ```
use CGI qw/:standard/;
```

```
Do stuff for ALL HTML pages here...
Setup header, page title, etc...

if (param()) {
 # User just submitted the form
 # process the data and put up a response page
} else {
 # User is surfing to the page for the
 # first time
 # show them a fill-in form/options menu/etc...
}
```

Slide 26

## Sticky Forms

- Sticky forms mean that the form fields are pre-populated with their previous values if you revisit them.
- The CGI.pm html shortcuts like `textfield()` provide this for free. Can be especially useful if you are going back to a form to ask a user to correct a mistake - they don't have to type everything again.

Slide 27

## Passing Data Between Pages

- Since HTTP is stateless and the Perl interpreter ends, you can't keep variables in memory across multiple page submissions.
- Two strategies:
  - Embed the data back into the page in hidden form fields. Simpler, if a bit clunky.
  - Store the data server-side in some persistent form. Harder, because you have to differentiate between multiple clients with a session key or similar.

Slide 28

## Client-Side Data Persistence

- Generate hidden form fields with CGI.pm's `hidden()` function. Read the perldoc.
- Safe and easy because each client then hands the data back in on the next request, so you know exactly who it is coming from.

Slide 29

## Server-Side CGI Data Persistence

- One way to do server-side persistence is to use CGI.pm's ability to set a cookie in the browser.
- Generate a unique ID, store it in a cookie, and then check for that cookie every time the script is hit.
- If cookie found, can then retrieve data previously stored on disk tagged with the unique key
- Requires user to accept cookies

Slide 30

## Some Notes on CGI

- The CGI script is called with a URL like `http://www.cis.uab.edu/fran/sample.cgi`
- The web server has to allow the `fran/` web directory to run CGI scripts
- The web server must be configured to recognize the `.cgi` extension as a CGI script
- `sample.cgi` must be set to be executable by the user running the web server (mode 755)

Slide 31

## More CGI.pm Usage Notes

- Make sure you speak HTML. Browsers don't understand `\n`, they understand `<br>`
- Always print out a proper header (and only one). It's generally appropriate to do this near the top of your script, not inside each case.

Slide 32

## Debugging CGI.pm Scripts

- If you run them from the command-line, they allow you to pass name=value pairs to represent the incoming data.
- Hit Ctrl-D when you are done setting input data, and the script will run. You'll see what it would have sent to the browser.

Slide 33

## More Debugging

- use `CGI::Carp qw(fatalsToBrowser);`
- This will redirect errors to the browser for easier debugging if you do not have access to the web server's error log.
- Read the `CGI::Carp` documentation for ways to redirect the errors to a file instead.

Slide 34

## Learn More About CGI

- `perldoc CGI` for the `CGI.pm` documentation
- Lincoln Stein's `CGI.pm` book - only of the only books published specifically for one Perl module.
- The Perl Cookbook has a chapter on CGI programming

Slide 35

## Simple Databases

- Three levels of databases...
  - Plain text files in some user-defined format
  - File-based databases like BerkeleyDB (DB)
  - SQL engines like MySQL, Oracle, MS-SQL, etc...
- You already know how to implement the first
- We will talk about the 2<sup>nd</sup> and 3<sup>rd</sup> methods

Slide 36

## DBM

- DBM actually refers to a variety of file-based database formats (NDBM, DB, GDBM, SDBM, ODBM)
- Different platforms had different DBM implementations so Perl needed multiple interfaces to support all of them.
- Perl ships with SDBM so that's guaranteed to be everywhere, so there is always some form of DBM available.

Slide 37

## DBM Abstraction

- All of these different implementations made it difficult to write portable programs
- Perl now has AnyDBM\_File, which will use whatever is available locally.
- The general philosophy of DBM files is that you open them, work on them just like a hash, and then save them to disk when done. They can then be retrieved during a later run.

Slide 38

## DBM Example

```
• use AnyDBM_File;
 dbmopen(my %stocks,"ticker",0666);
 print "Enter a stock ticker symbol: ";
 chomp(my $sym = <STDIN>);
 print "Enter a price for $sym: ";
 chomp(my $price = <STDIN>);
 $stocks{$sym} = $price;
 for (keys %stocks) {
 print "$_ is $stocks{$_}.\n";
 }
 dbmclose(%stocks);
```

- Each run through this will add another stock symbol and price to the DBM

Slide 39

## DBM Limitations

- DBMs are designed to store simple key=value pairs. Most can't store references to data more complex than scalars (they'll actually store the string "HASH{0x2bbc4a}" instead)
- Solution: make multiple DBMs, or use MLDBM. It uses Data::Dumper to generate storable strings from complex data. It's not standard, though.
- DBMs also have record size limits (1k is safe)

Slide 40

## Emptying a DBM File

- Quite easy....
- ```
dbmopen (my %stocks, 'ticker', 0666);  
%stocks = ();  
dbmclose(%stocks);
```

Slide 41

Sorting Large DBM Files

- It's worth noting that DB_File (BerkeleyDB) has a BTREE mode which will store your data as a binary tree. This is much more efficient for large data sets.
- Now, calls to keys(), values() and each() come back ordered automatically.
- You can provide a comparison function to tell DB_File how to sort the data.

Slide 42

SQL Databases - DBI

- Perl provides an extremely robust abstraction layer called the DBI which allows you to write database-independent code to execute SQL queries.
- With the DBI, it is very easy to port applications between databases, and the syntax remains the same no matter which database you are using.

Slide 43

DBI Example

- ```
use DBI;
my $dbh = DBI->connect('DBI:driver:database',
 'username',
 'auth',
 {RaiseError => 1,
 AutoCommit => 1, });

$dbh->do($sql);
my $sth = $dbh->prepare($sql);
$sth->execute();
while (my @row = $sth->fetchrow_array) {
 # do something
}
$sth->finish();
$dbh->disconnect();
```

Slide 44

## DBD

- The DBI works by sitting on top of a database-specific DBD driver library. There are DBD modules available for just about all popular databases, including MySQL, PostgreSQL, Oracle, and ODBC (for MS and other databases).
- Specify in the connect string:  
my \$dbh = DBI->connect('DBI:mysql:dbname'...