Slide 1

The Long and the Short of Perl

Marty Pauley <marty@kasei.com>

火星

JOIN '04

Slide 2

Preface

"scripting language" often sounds like an insult, especially when said by some C++, C#, and Java programmers. They imply that *scripting* languages are limited in some way.

For languages like Perl, Python, Ruby, and TCL, "high-level language" is a better description. These languages are much more expressive than low-level languages like C++, C# and Java.

Contents

1 What is Perl? 3

2 Long Perl 13
2.1 Object orientation . . . 14
2.2 Other paradigms . . . 16
2.3 Higher-level programming . . . 17
2.4 Lexical closures . . . 18
2.5 Symbol table access . . . 19

3 Short Perl 21
3.1 Running Perl . . . 22
3.2 cat . . . 23
3.3 **** off, cat! . . . 25

4 CPAN 27
4.1 A simple class . . . 28
4.2 Database access . . . 32
4.3 CGI script . . . 35
4.4 WWW client . . . 39
4.5 Email . . . 41
4.6 Box, Ox, Octopus, and Sheep . . . 43
4.7 Inline . . . 44
4.8 Other useful modules . . . 45

5 More information 46
5.1 YAPC::Europe . . . 47

Slide 3

1 What is Perl?

`perldoc perlintro`

Perl is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, web development, network programming, GUI development, and more.

...

Slide 4

What is Perl?

`perldoc perlintro`

The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal). Its major features are that it's easy to use, supports both procedural and object-oriented (OO) programming, has powerful built-in support for text processing, and has one of the world's most impressive collections of third-party modules.

...

Slide 5

What is Perl?

```
perldoc perlintro
```

Different definitions of Perl are given in *perl*, *perlfaq1* and no doubt other places. From this we can determine that Perl is different things to different people, but that lots of people think it's at least worth writing about.

Slide 6

What is Perl?

```
perldoc perlfaq1
```

Perl is a high-level programming language with an eclectic heritage written by Larry Wall and a cast of thousands. It derives from the ubiquitous C programming language and to a lesser extent from sed, awk, the Unix shell, and at least a dozen other tools and languages. Perl's process, file, and text manipulation facilities make it particularly well-suited for tasks involving . . .

Slide 7

What is Perl?

. . . quick prototyping, system utilities, software tools, system management tasks, database access, graphical programming, networking, and world wide web programming. These strengths make it especially popular with system administrators and CGI script authors, but mathematicians, geneticists, journalists, and even managers also use Perl. **Maybe you should, too.**

Slide 8

What is Perl?

If you want to serve roast meat, use a carving knife and fork.

Slide 9

What is Perl?

If you want pretty nails, use a nail file.

Slide 10

What is Perl?

If you want to fix your bike, use a screwdriver.

Slide 11

What is Perl?

If you want to kill your enemies, a Samurai sword might be useful.

Slide 12

What is Perl?

For all of the above. . .

Slide 13

2 Long Perl

Some people think Perl can't be used for large systems.

Sadly, some of these people actually work as Perl programmers.

But they're wrong! Perl is an excellent language for large and complex systems.

Slide 14

2.1 Object orientation

Object orientation is still the current paradigm of choice for writing large programs; Perl supports OO.

Normal Perl OO is class-based and allows multiple inheritance.

Perl actually allows you to use many different forms of object orientation, if you want.

Slide 15

Object orientation

Classes in Perl are not closed. It's easy to add a new method to an existing class, without even modifying the module file.

You can even add methods to the UNIVERSAL class.

```
sub UNIVERSAL::moniker {
    (ref( $_[0] ) || $_[0]) =~ /([^:]+)$/;
    return lc $1;
}
```

Slide 16

2.2 Other paradigms

Perl supports object orientation but **doesn't force you to use it**.

Perl also supports procedural and functional paradigms. You can choose whatever style suits your problem.

Slide 17

2.3 Higher-level programming

Perl allows you to write code that writes code.

The most obvious way is to use the `eval` function, but there are even more elegant ways.

Slide 18

2.4 Lexical closures

A lexical closure is a subroutine that refers to variables in the lexical scope where it was created. It's surprising how useful such a simple construct can be.

```
sub iterator {
    my @data = @_;
    return sub { shift @data }
}
```

Slide 19

2.5 Symbol table access

Perl package namespaces can be accessed from within the program: that's how the `import` subroutine works. You can examine and modify namespaces easily.

```
no warnings 'redefine';
local *MIME::Lite::send = sub {
    warn "Sending email\n" };
```

That can be useful while testing.

Slide 20

Symbol table access, and closures

```
sub trace {
    my $victim = shift;
    no strict 'refs';
    no warnings 'redefine';
    my $orig = *{$victim}{CODE};
    *{$victim} = sub {
        warn "calling $victim(@_)\n";
        goto $orig; };
}
trace('MIME::Lite::send');
```

Slide 21

3 Short Perl

Perl is an excellent language for *short* programs.

Perl doesn't force you to write "scaffolding" code.

Perl will even write some code for you, if you ask.

Slide 22

3.1 Running Perl

We know that Perl code can live in files, but it doesn't have to. You can run short Perl programs directly from the command line by using the `-e` switch.

```
perl -e 'print "Hello World\n"'
```

Useful Perl programs can be so short that you can easily write them each time you need them.

Slide 23

3.2 cat

Implementing the Unix `cat` program in Perl is not difficult.

```
perl -e 'while (<>) {print}' *.txt
```

But you can shorten that.

```
perl -ne 'print' *.txt
```

The `-n` switch wraps your code in an input loop, so the running code is the same in both examples above.

Slide 24

Cheshire cat

If that wasn't short enough, you can remove all your code.

```
perl -pe '' *.txt
```

The `-p` option wraps your code in an input loop like `-n`, but also adds a `print` in every iteration.

```
while (<>) {} continue {print}
```

Slide 25

3.3 **** off, cat!

There is no good reason to reimplement `cat` in Perl, unless you're using a Windows machine. But you might want a *smart* `cat` that will replace all the offensive words with asterisks.

```
perl -MRegexp::Common -pe
    's/$RE{profanity}/****/g' *.txt
```

Slide 26

Keeping it clean

You might want to permanently remove those offensive words from your files. Perl makes that easy too.

```
perl -MRegexp::Common -i -pe
    's/$RE{profanity}/****/g' *.txt
```

The `-i` option performs *in-place* editing.

Slide 27

4 CPAN

The Comprehensize Perl Archive Network is a repository of modules for every task you can imagine, and then some more!

Have a look at `http://search.cpan.org/`

You don't need to write the code if someone else already has.

CPAN is why *you* should use Perl.

Slide 28

4.1 A simple class

In many other OO languages even simple classes require more code than they should.

A simple class example that is frequently used is an "employee" class. An employee has attributes for his name, tax reference, and annual salary.

Slide 29

A simple class

With `Class::Accessor` from CPAN we could write:

```
package Employee;
use base 'Class::Accessor';
__PACKAGE__->mk_accessors(
    qw(name taxref salary));
```

Slide 30

A simple class

Then you can do

```
my $worker = Employee->new({
    name => 'Fred', taxref => 'N1234B' });
$worker->salary(20000);

print $worker->name, " earns ",
    $worker->salary, " euro\n";
```

Slide 31

A simple class

Alternativly, if your Employee class isn't going to get any more complicated, you could use `Class::Struct` instead:

```
use Class::Struct;
struct Employee => { name => '$',
    taxref => '$', salary => '$' };
```

Slide 32

4.2 Database access

We have a MySQL movie database containing a "film" table with information about specific films, like the title, genre, year of release, and plot outline.

We need to implement a `Film` class that will allow us to

- search for specific films
- update the information for a film
- add new films
- remove existing films

Slide 33

Database access

Our *complete* film class would look like this:

```
package Film;
use base 'Class::DBI::mysql';
__PACKAGE__->set_db('Main',
    'dbi:mysql:movies',
    'user', 'password');
__PACKAGE__->set_up_table("film");
```

Slide 34

Database access

Then you can do

```
my $x2 = Film->create({
    title => 'X-Men 2',
    genre => 'sci-fi'});

$x2->plot('good mutants fight baddies');
$x2->update;

my @sf = Film->search(genre => 'sci-fi');
```

Slide 35

4.3 CGI script

Now that we have a Film class, it would be great to have a simple CGI script to display a list of films. When a film is selected from the list, the script should display all the details for that film.

Slide 36

CGI script

```
use CGI ':standard';
print header;
if (my $id = param('id')) {
    show_film($id);
} else {
    show_list();
}
```

Slide 37

CGI script

```
sub show_film {
    my $film = Film->retrieve($_[0]);
    print start_html($film->title),
        h1($film->title), p($film->plot),
        p($film->genre, $film->release),
        end_html;
}
```

Slide 38

CGI script

```
sub show_list {
    print start_html("films"),
        h1("all films"),
        ul( map li(
            a({href=>"?id=$_"}, $_->title)),
            Film->retrieve_all),
        end_html;
}
```

Slide 39

4.4 WWW client

It would be great if we could use our offensive word elimator on the WWW.

```
use LWP::Simple;
use Regexp::Common;
my $page = get($url);
$page =~ s/$RE{profanity}/****/g;
print $page;
```

Slide 40

Another WWW client

Perl doesn't enforce any particular moral standards...

```
use Acme::PrOn::Automate qw(:categories);
my $naughty = Acme::PrOn::Automate->new(
    sources => [qw(Free6 Easypic)],
    categories => [BABES, LINGERIE],
    db => "naughty_db",
);
$naughty->fetch();
```

Slide 41

4.5 Email

Back to our quest to remove offensive words. This time, let's clean up our email. We can start by filtering offensive email.

```
use Mail::Audit;
use Regexp::Common;
my $msg = Mail::Audit->new();
if (grep /$RE{profanity}/, @{$msg->body}) {
    $mail->reject("offensive email");
}
$msg->accept("inbox");
```

Slide 42

Email

Now check email received before we started filtering.

```
use Mail::Box::Manager;
use Regexp::Common;
my $mgr = Mail::Box::Manager->new;
my $box = $mgr->open(folder => $ENV{MAIL});
for my $msg ($box->messages) {
    $msg->delete
        if $msg->string =~ /$RE{profanity}/;
}
$box->close;
```

Slide 43

4.6 Box, Ox, Octopus, and Sheep

English has some obscure rules to make plurals, and Perl knows them.

```
use Lingua::EN::Inflect 'PL';
for (qw/box ox octopus sheep/) {
    print "$_ => ", PL($_), "\n";
}
```

Slide 44

4.7 Inline

"I have to use a particular Java class."

Again, Perl doesn't enforce any moral standards.

```
package Searcher;
use Inline Java => 'STUDY',
    STUDY => [qw( com.kasei.Lucy )],
    SHARED_JVM => 1;
my $jc = Searcher::com::kasei::Lucy->new;
$jc->search("scripting language");
```

Slide 45

4.8 Other useful modules

- `AI::NeuralNet::Simple`
- `Algorithm::Diff`
- `Bioperl`
- `File::Slurp`
- `POE`
- `Spreadsheet::WriteExcel::Simple`
- `Template` Toolkit

Slide 46

5 More information

Perl comes with lots of documentation. `perldoc perl` will give you a list of other documents to read, both tutorial and reference.

`http://perl.com/` sometimes has good articles.

`http://perl.org/` has links to a wide range of Perl stuff.

Slide 47

5.1 YAPC::Europe

Yet Another Perl Conference for Europe will be in Belfast in September 2004.

Many of the best Perl Hackers will be there.

Admission is 99 Euro for 3 days.

`http://belfast.yapc.org/` for more details.