

From CGI to mod_perl 2.0, Fast!

Philippe M. Chiasson
gozer@ectoplasm.org



Fast!

```
$> wget http://perl.apache.org/download/mod_perl-2.0-current.tar.gz
$> tar zxvf mod_perl-2.0-current.tar.gz
[...]
$> cd mod_perl-2.0.*/
$> perl Makefile.PL MP_APXS=`which apxs`
Configuring Apache/2.0.52 mod_perl2/2.0.2 Perl/v5.8.6
[...]
$> make
cd "src/modules/perl" && make
[...]
$> make test

$> make install
$> perl -pi -e's/cgi_module/perl_module/g' `apxs -q SYSCONFDIR`/httpd.conf
$> perl -pi -e's/mod_cgi/mod_perl/g' `apxs -q SYSCONFDIR`/httpd.conf
$> perl -pi -e's/cgi-script/perl-script/g' `apxs -q SYSCONFDIR`/httpd.conf
$> `apxs -q SBINDIR`/apachectl configtest
Syntax OK
$> `apxs -q SBINDIR`/apachectl restart
```

Faster!

```
$> wget -O- http://gozer.ectoplasm.org/talks/ApacheCon/2005/US/mod\_perl-2.0-cgi-fast/fast.sh | sh -x
```

Better!

```
$> yum install mod_perl
```

```
$> apt-get mod_perl
```

```
$> fink install mod_perl
```

```
$> pkg_add -r mod_perl
```

```
$> emerge mod_perl
```

Thank You!

Do we still have time?

Let's get serious!

You have

- Apache 2.x
- mod_cgi
- A bunch of CGI scripts
- A working web site
- A good job

Common Gateway Interface

- CGI is simple
- CGI is easy
- CGI just works

How about mod_perl?

- mod_perl is simple
- mod_perl is easy
- mod_perl just works

How about mod_perl?

- mod_perl is simple
- mod_perl is easy
- mod_perl just works **BETTER**

How about mod_perl?

- mod_perl is simple
- mod_perl is easy
- mod_perl just works **FASTER**

How about mod_perl?

- mod_perl is simple
- mod_perl is easy
- mod_perl just works EASIERer

But why switch?

- CPU cycles
- Memory
- Servers
- Apache::* module on CPAN

The CGI model

- Forking
- Startup
- Teardown

The CGI model

- `httpd fork()`s
- `httpd exec(/var/www/cgi-bin/myscript.pl)`
- OS find `#!/usr/bin/perl` in `/var/www/cgi-bin/myscript.pl`
- OS `fork()`s
- OS `exec()`s `/usr/bin/perl`
- Perl opens, initializes and parses `myscript.pl`
- Perl runs the script
- Perl shutdowns

The CGI model

2x fork()
2x exec()
Perl startup
Perl shutdown

EXPENSIVE

The mod_perl model

- Embedded Perl
- One time startup
- One time teardown

The mod_perl model

- Perl locates the correct subroutine to invoke
- Perl calls it

CHEAP_{er}

Getting there from here

mod_cgi setup

```
LoadModule cgi_module libexec/mod_cgi.so
```

```
ScriptAlias /cgi-bin /var/www/cgi-bin
```

mod_cgi setup

ScriptAlias is just shorthand

```
LoadModule cgi_module libexec/mod_cgi.so
```

```
Alias /cgi-bin /var/www/cgi-bin
```

```
<Location /cgi-bin>
```

```
    SetHandler cgi-script
```

```
    Option +ExecCGI
```

```
</Location>
```

mod_cgi setup

`/var/www/cgi-bin/hello.pl`

```
#!/usr/bin/perl

use CGI;

my $q = new CGI;

print $q->header('text/html');

print <<"EOF";
<html><body>
<h1>Hello world!</h1>
<pre>
GATEWAY_INTERFACE: $ENV{GATEWAY_INTERFACE}
MOD_PERL: $ENV{MOD_PERL}
</pre>
</body></html>
EOF
```

mod_cgi setup



How fast is that ?

- Apache Bench (ab)
- Nice benchmarking tool
- Will work for us nicely
- Comes with Apache

How fast is that ?

```
$> ab -c1 -n50 http://localhost:8529/cgi-bin/hello.pl  
Benchmarking 127.0.0.1 (be patient).....done  
Requests per second:      5.57 [# /sec] (mean)  
Time per request:        179.448 [ms] (mean)  
Transfer rate:           1.67 [Kbytes/sec] received
```

mod_perl time

```
$> wget http://perl.apache.org/download/mod_perl-2.0-current.tar.gz
$> tar zxvf mod_perl-2.0-current.tar.gz
[...]
$> cd mod_perl-2.0.*/
$> perl Makefile.PL MP_APXS=`which apxs`
Configuring Apache/2.0.52 mod_perl2/2.0.2 Perl/v5.8.6
[...]
$> make
cd "src/modules/perl" && make
[...]
$> make test

$> make install
$> perl -pi -e's/cgi_module/perl_module/g' `apxs -q SYSCONFDIR`/httpd.conf
$> perl -pi -e's/mod_cgi/mod_perl/g' `apxs -q SYSCONFDIR`/httpd.conf
$> perl -pi -e's/cgi-script/perl-script/g' `apxs -q SYSCONFDIR`/httpd.conf
$> `apxs -q SBINDIR`/apachectl configtest
Syntax OK
$> `apxs -q SBINDIR`/apachectl restart
```

mod_perl time

Download & unpack

```
$> wget http://perl.apache.org/download/mod\_perl-2.0-current.tar.gz
$> tar zxvf mod_perl-2.0-current.tar.gz
[...]
$> cd mod_perl-2.0.*/
```

mod_perl time

Configure

```
$> /usr/bin/perl Makefile.PL MP_APXS=/usr/sbin/apxs  
Configuring Apache/2.0.52 mod_perl2/2.0.2 Perl/v5.8.6  
[...]
```

mod_perl time

Build, test, install

```
$> make  
$> make test  
$> make install
```

mod_perl time

Check your distro!

mod_perl is supported by lots of them

mod_perl time

httpd.conf

```
LoadModule cgi_module modules/mod_cgi.so  
LoadModule perl_module modules/mod_perl.so
```

```
Alias /cgi-bin /var/www/cgi-bin  
Alias /perlrun /var/www/cgi-bin
```

```
<Location /cgi-bin>  
  SetHandler cgi-script  
  Option +ExecCGI  
</Location>  
<Location /perlrun>  
  SetHandler perl-script  
  Option +ExecCGI  
  PerlHandler ModPerl::PerlRun  
</Location>
```


mod_perl time

- SetHandler
- PerlHandler
- ModPerl::PerlRun

mod_perl time

```
$> apachectl restart  
[notice] Apache/2.0.52 (Unix) mod_perl/2.0.2-dev Perl/v5.8.7 configured
```

mod_perl time



it works!

- `$ENV{MOD_PERL} = "mod_perl/2.0.2-dev";`

How fast is THAT ?

```
$> ab -c1 -n50 http://localhost:8529/perlrun/hello.pl  
Benchmarking 127.0.0.1 (be patient).....done  
Requests per second:      83.15 [#/sec] (mean)  
Time per request:        12.027 [ms] (mean)  
Transfer rate:           26.61 [Kbytes/sec] received
```

How fast is THAT ?

mod_cgi

```
$> ab -c1 -n50 http://localhost:8529/cgi-bin/hello.pl  
Benchmarking 127.0.0.1 (be patient).....done  
Requests per second:      5.57 [#/sec] (mean)  
Time per request:        179.448 [ms] (mean)  
Transfer rate:           1.67 [Kbytes/sec] received
```

ModPerl::PerlRun

```
$> ab -c1 -n50 http://localhost:8529/perlrun/hello.pl  
Benchmarking 127.0.0.1 (be patient).....done  
Requests per second:      83.15 [#/sec] (mean)  
Time per request:        12.027 [ms] (mean)  
Transfer rate:           26.61 [Kbytes/sec] received
```

How fast is THAT ?

mod_cgi

```
$> ab -c1 -n50 http://localhost:8529/cgi-bin/hello.pl  
Benchmarking 127.0.0.1 (be patient).....done  
Requests per second:      5.57 [#/sec] (mean)  
Time per request:        179.448 [ms] (mean)  
Transfer rate:           1.67 [Kbytes/sec] received
```

ModPerl::PerlRun

```
$> ab -c1 -n50 http://localhost:8529/perlrun/hello.pl  
Benchmarking 127.0.0.1 (be patient).....done  
Requests per second:      15x(mod_cgi) [#/sec] (mean)  
Time per request:        15x(mod_cgi) [ms] (mean)  
Transfer rate:           16x(mod_cgi) [Kbytes/sec] received
```

ModPerl::PerlRun

- mod_perl's bundled module
- closest CGI emulation available

ModPerl::PerlRun

On every request for the script:

- Found
- Loaded
- Parsed
- BEGIN {
- Executed
- END }
- Destroyed

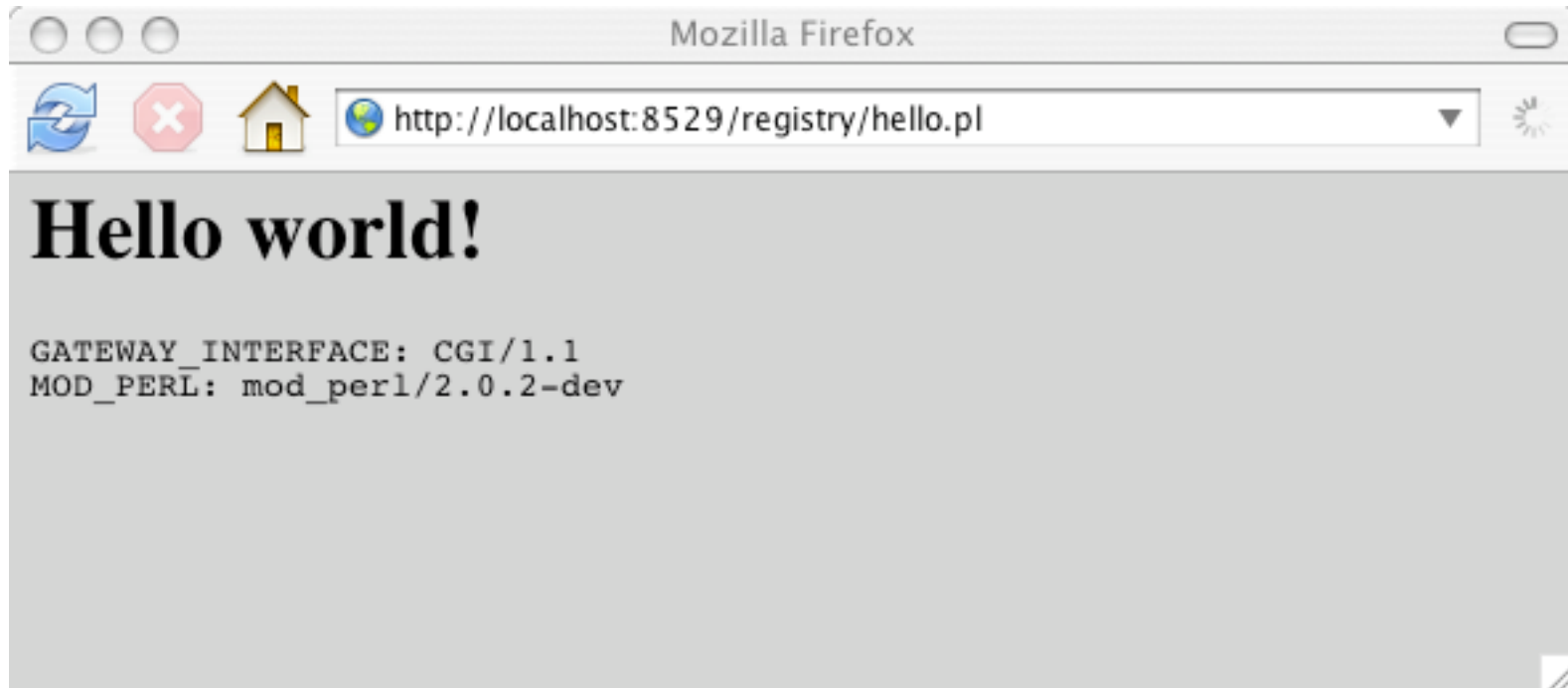
ModPerl::Registry

httpd.conf

```
Alias /cgi-bin /var/www/cgi-bin
Alias /perlrun /var/www/cgi-bin
Alias /registry /var/www/cgi-bin
```

```
<Location /cgi-bin>
  SetHandler cgi-script
  Option +ExecCGI
</Location>
<Location /perlrun>
  SetHandler perl-script
  Option +ExecCGI
  PerlHandler ModPerl::PerlRun
</Location>
<Location /registry>
  SetHandler perl-script
  Option +ExecCGI
  PerlHandler ModPerl::Registry
</Location>
```

ModPerl::Registry



How fast is THAT ?

```
$> ab -c1 -n50 http://localhost:8529/registry/hello.pl  
Benchmarking 127.0.0.1 (be patient).....done  
Requests per second:      156.37 [# /sec] (mean)  
Time per request:         6.395 [ms] (mean)  
Transfer rate:            50.04 [Kbytes/sec] received
```

How fast is THAT ?

mod_cgi

```
$> ab -c1 -n50 http://localhost:8529/cgi-bin/hello.pl
Benchmarking 127.0.0.1 (be patient).....done
Requests per second:      5.57 [#/sec] (mean)
Time per request:        179.448 [ms] (mean)
Transfer rate:           1.67 [Kbytes/sec] received
```

ModPerl::PerlRun

```
$> ab -c1 -n50 http://localhost:8529/perlrun/hello.pl
Benchmarking 127.0.0.1 (be patient).....done
Requests per second:      83.15 [#/sec] (mean)
Time per request:        12.027 [ms] (mean)
Transfer rate:           26.61 [Kbytes/sec] received
```

ModPerl::Registry

```
$> ab -c1 -n50 http://localhost:8529/registry/hello.pl
Benchmarking 127.0.0.1 (be patient).....done
Requests per second:      156.37 [#/sec] (mean)
Time per request:         6.395 [ms] (mean)
Transfer rate:           50.04 [Kbytes/sec] received
```

How fast is THAT ?

mod_cgi

```
$> ab -c1 -n50 http://localhost:8529/cgi-bin/hello.pl
Benchmarking 127.0.0.1 (be patient).....done
Requests per second:      5.57 [#/sec] (mean)
Time per request:        179.448 [ms] (mean)
Transfer rate:           1.67 [Kbytes/sec] received
```

ModPerl::PerlRun

```
$> ab -c1 -n50 http://localhost:8529/perlrun/hello.pl
Benchmarking 127.0.0.1 (be patient).....done
Requests per second:      15x(mod_cgi) [#/sec] (mean)
Time per request:        15x(mod_cgi) [ms] (mean)
Transfer rate:           16x(mod_cgi) [Kbytes/sec] received
```

ModPerl::Registry

```
$> ab -c1 -n50 http://localhost:8529/registry/hello.pl
Benchmarking 127.0.0.1 (be patient).....done
Requests per second:      28x(mod_cgi) [#/sec] (mean)
Time per request:        28x(mod_cgi) [ms] (mean)
Transfer rate:           30x(mod_cgi) [Kbytes/sec] received
```

ModPerl::Registry

- mod_perl's bundled module
- Faster, but CGI emulation lacking

ModPerl::Registry

On every request for the script:

- Load from file if:
 - Not found in cache
 - Cache entry is older than the file
- BEGIN {
- Executed
- END }

ModPerl::(PerlRun|Registry)

- Try ModPerl::Registry first
 - faster
 - might break some CGIs
- Try ModPerl::PerlRun after
 - slower
 - might break some CGIs

ModPerl::(PerlRun|Registry)

- Simple Perl modules
- Designed to run perl scripts from mod_perl pretending it's a CGI
- Derives from ModPerl::RegistryCooker
- New ones can be made in a pinch

```

package ModPerl::PerlRun;
use base qw(ModPerl::RegistryCooker);

sub handler : method {
    my $class = (@_ >= 2) ? shift : __PACKAGE__;
    my $r = shift;
    return $class->new($r)->default_handler();
}

my $parent = 'ModPerl::RegistryCooker';
my %aliases = (
    new          => 'new',
    init         => 'init',
    default_handler => 'default_handler',
    run          => 'run',
    can_compile  => 'can_compile',
    make_namespace => 'make_namespace',
    namespace_root => 'namespace_root',
    namespace_from => 'namespace_from_filename',
    is_cached    => 'FALSE',
    should_compile => 'TRUE',
    flush_namespace => 'flush_namespace_normal',
    cache_table  => 'cache_table_common',
    cache_it     => 'NOP',
    read_script  => 'read_script',
    shebang_to_perl => 'shebang_to_perl',
    get_script_name => 'get_script_name',
    chdir_file   => 'NOP',
    get_mark_line => 'get_mark_line',
    compile      => 'compile',
    error_check  => 'error_check',
    should_reset_inc_hash => 'TRUE',
    strip_end_data_segment => 'strip_end_data_segment',
    convert_script_to_compiled_handler =>
'convert_script_to_compiled_handler',
);

$aliases{$_} = $parent . "::" . $aliases{$_} for keys %aliases;
__PACKAGE__->install_aliases(\%aliases);

```

CGI emulation

- It's an emulation
- It ain't the real thing
- things can go wonky

I remember

```
#!/usr/bin/perl

use CGI;

my $q = new CGI;

print $q->header('text/plain');

$count++;
```



```
print <<"EOF";
counted $count
EOF
```

I remember

```
#!/usr/bin/perl
```

```
use CGI;
```

```
my $q = new CGI;
```

```
print $q->header('text/plain');
```

```
$counter++;
```

```
print <<"EOF";  
counted $counter  
EOF
```

```
$> GET http://localhost:8529/cgi-bin/count.pl  
counted 1
```

```
$> GET http://localhost:8529/cgi-bin/count.pl  
counted 1
```

```
$> GET http://localhost:8529/cgi-bin/count.pl  
counted 1
```

I remember

```
#!/usr/bin/perl
```

```
use CGI;
```

```
my $q = new CGI;
```

```
print $q->header('text/plain');
```

```
$counter++;
```

```
print <<"EOF";  
counted $counter  
EOF
```

```
$> GET http://localhost:8529/registry/count.pl
```

```
counted 1
```

```
$> GET http://localhost:8529/registry/count.pl
```

```
counted 2
```

```
$> GET http://localhost:8529registry/count.pl
```

```
counted 3
```

```
$> GET http://localhost:8529registry/count.pl
```

```
counted 1
```

```
$> GET http://localhost:8529registry/count.pl
```

```
counted 4
```

That can't be right?

```
#!/usr/bin/perl

use CGI;

my $q = new CGI;

print $q->header('text/plain');

$counter++;

print <<"EOF";
counted $counter
EOF
```


That can't be right?

- Code is cached
- Perl sticks around
- Great for speed
- Globals are suddenly **VERY GLOBAL**

That can't be right?

```
#!/usr/bin/perl

use CGI;

my $q = new CGI;

print $q->header('text/plain');

$counter++;

print <<"EOF";
counted $counter
EOF
```

That can't be right?

```
#!/usr/bin/perl

use CGI;

my $q = new CGI;

print $q->header('text/plain');

$main::counter++;    # GLOBAL!

print <<"EOF";
counted $counter
EOF
```

That can be right!

```
#!/usr/bin/perl

use CGI;

my $counter;
my $q = new CGI;

print $q->header('text/plain');

$counter++;

print <<"EOF";
counted $counter
EOF
```

Globals will stick

- Globals will stick around
- Use my
 - Try `ModPerl::PerlRun`

CGI emulation

- It's an emulation
- It ain't the real thing
- things can go wonky

I still remember

```
#!/usr/bin/perl

use CGI;
use strict;
use warnings;

my $q = new CGI;
my $counter;

print $q->header('text/plain');

counter_up();

sub counter_up {
    $counter++;
    print "Counter: $counter\n";
}
```

```
$> GET http://localhost:8529/registry/count.pl
Counter: 1
$> GET http://localhost:8529/registry/count.pl
Counter: 2
$> GET http://localhost:8529/registry/count.pl
Counter: 3
$> GET http://localhost:8529/registry/count.pl
Counter: 1
$> GET http://localhost:8529/registry/count.pl
Counter: 4
```

But, what the..?

- I used my so it's not a global...
- To Perl, all code lives in subroutines
- mod_perl turns scripts into subroutines
- subroutines can create closures

subification*

```
#!/usr/bin/perl

use CGI;
use strict;
use warnings;

my $q = new CGI;
my $counter;

print $q->header('text/plain');

counter_up();

sub counter_up {
    $counter++;
    print "Counter: $counter\n";
}
```

* the process of making a subroutine, obviously

subification^{*}

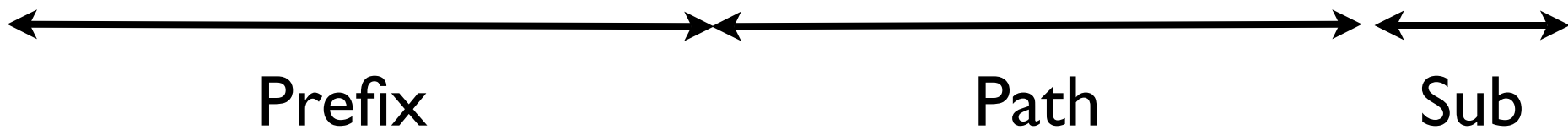
- subroutine needs a name
- filepath is a convenient naming

* the process of making a subroutine, obviously

subification*

/var/www/cgi-bin/hello.pl

`ModPerl::ROOT::ModPerl::Registry::var_www_cgi_2dbin_hello_2ep1::handler`



* the process of making a subroutine, obviously

subification*

sub

```
ModPerl::ROOT::ModPerl::Registry::var_www_cgi_2dbin_count_2epl::handler {  
package ModPerl::ROOT::ModPerl::Registry::var_www_cgi_2dbin_count_2epl;  
no warnings;  
local $0 = '/var/www/cgi-bin/count.pl';
```

```
use CGI;  
use strict;  
use warnings;
```

```
my $q = new CGI;  
my $counter;
```

```
print $q->header('text/plain');
```

```
counter_up();
```

```
sub counter_up {  
    $counter++;  
    print "Counter: $counter\n";  
}
```

```
}
```

* the process of making a subroutine, obviously

closures

- Closures are a Perl feature
- Come to be when creating subs
- Normally not an issue
- `mod_perl` can create some without telling you
- `man perlref` for more

I still remember

```
#!/usr/bin/perl
```

```
use CGI;  
use strict;  
use warnings;
```

```
my $q = new CGI;  
my $counter;
```

```
print $q->header('text/plain');  
  
counter_up();
```

```
sub counter_up {  
    $counter++;    # LEXICAL/CLOSURE  
    print "Counter: $counter\n";  
}
```

2 warning signs to look for:
globals

error_log:

```
Variable "$counter" will not stay  
shared at /var/www/cgi-bin/  
count.pl
```

I still remember

```
#!/usr/bin/perl

use CGI;
use strict;
use warnings;

my $q = new CGI;
my $counter;

print $q->header('text/plain');

counter_up($counter);

sub counter_up {
    my $counter = shift;
    $counter++;    # GOOD
    print "Counter: $counter\n";
}
```

Just remember that
globals in general are to
watch out for

Good advice

- use strict;
- use warnings;
- avoid globals
 - Unless they are read-only
- look for hints in your error_log file

CGI emulation

- It's an emulation
- It ain't the real thing
- things can go wonky

I used to work

```
#!/usr/bin/perl
```

```
use CGI;  
use strict;  
use warnings;
```

```
require "countlib.pl";
```

```
my $q = new CGI;
```

```
print $q->header('text/plain');
```

```
counter_up();
```

```
#countlib.pl
```

```
use strict;  
my $counter = 0;
```

```
sub counter_up {  
    $counter++;  
    print "Counter: $counter\n";  
}
```

I used to work



error_log:

```
[Sun Oct 23 21:37:44 2005] [error] Can't locate countlib.pl in @INC (@INC contains: [...]) at /var/www/cgi-bin/hello.pl line 7.\n
```

I used to work

```
#!/usr/bin/perl
```

```
use CGI;  
use strict;  
use warnings;  
use Cwd;
```

```
my $q = new CGI;
```

```
print $q->header('text/plain');  
print "Cwd: ", cwd(), "\n";
```

```
$> GET http://localhost:8529/cgi-bin/cwd.pl  
Cwd: /var/www/cgi-bin  
$> GET http://localhost:8529/registry/cwd.pl  
Cwd: /
```

I used to work

- `mod_cgi`
 - `fork()`s
 - `chdir()`s to the script's directory

Cwd

- mod_perl
 - doesn't fork()
 - doesn't chdir()s
 - could chdir()s

Cwd

- Cwd() is a process property
- Apache has threaded MPMs
- chdir() isn't thread-safe
- mod_perl doesn't try to break things

Cwd

- You can fix your scripts
 - use `lib(/var/www/lib);`
 - require “/fully/qualified/lib/path.pl”;

Cwd

- If your MPM is Prefork:
 - `ModPerl::RegistryPrefork`
 - `ModPerl::PerlRunPrefork`

It ain't magical!

- use `strict/warnings`
- Think about globals
- watch the `error_log`

More info

- *mod_perl User's mailing-list*
 - <http://perl.apache.org/maillist/modperl.html>
 - [<modperl@perl.apache.org>](mailto:modperl@perl.apache.org)
- *mod_perl Developer's Cookbook*
 - <http://www.modperlcookbook.org/>
- **Practical mod_perl**
 - <http://www.modperlbook.org/>
- **mod_perl at the ASF**
 - <http://perl.apache.org/>

Thank You!

Slides and bonus material:

<http://gozer.ectoplasm.org/talk/>