B211 Internet Computing

CGI Programming with Perl

Learning Objectives

1. Learn how to access and use CGI scripts from the web browser.

2. Learn how to write simple CGI scripts using Perl by following some examples.

3. Understand the communications between the web client and web server during CGI communication.

Understanding Perl

- Don't worry too much about understanding the details of the Perl scripts I present in this lecture. They are mainly here so that you can understand some basic concepts in CGI.
  - We will be going through the syntax for Perl in the next lecture.
  - Also, I give the complete scripts here so that you can copy them to do your CGI programming exercises next week.

- You will get a picture of what Perl scripts looks like in this lecture. Return to them after you have learnt the syntax for Perl - they will make much more sense then.

Trying out the scripts

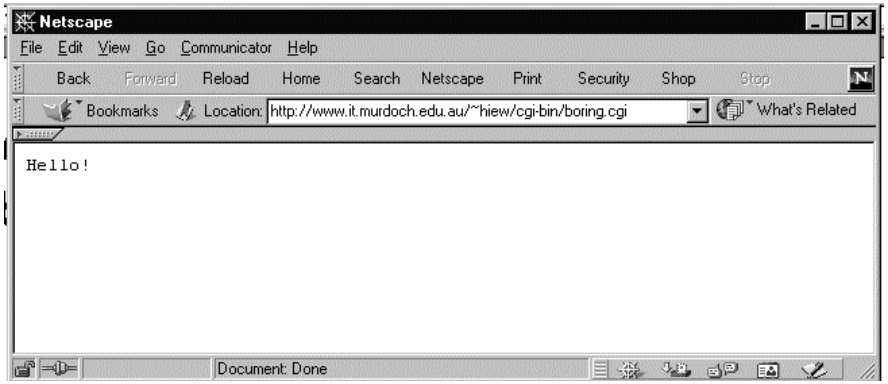
- While reading these lecture notes, start up your browser and try out the scripts as indicated by the screen dumps. All those scripts really do exist as indicated.

A simple CGI script

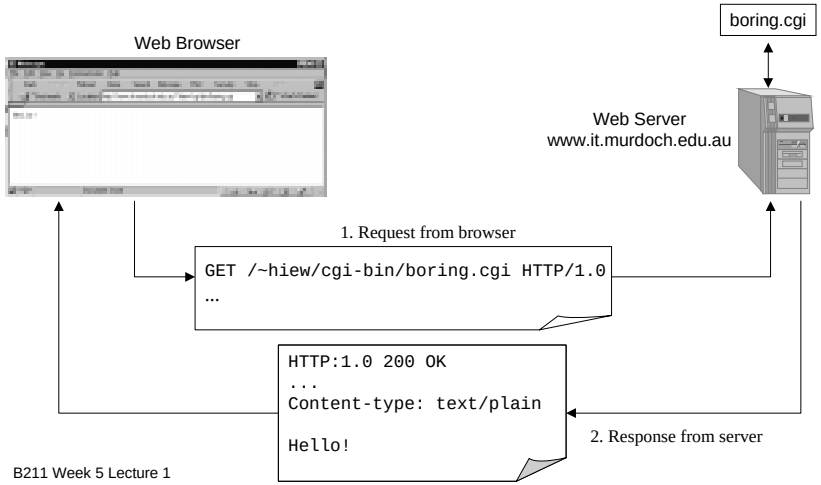
- Location
  - http://www.it.murdoch.edu.au/~hiew/cgi-bin/boring.cgi
- All it does is print the word “Hello!”

```
#!/usr/bin/perl -w

print "Content-type: text/plain\n\n" ;
print "Hello!" ;
```


Accessing the script from a browser



What actually happens:



A more interesting CGI script

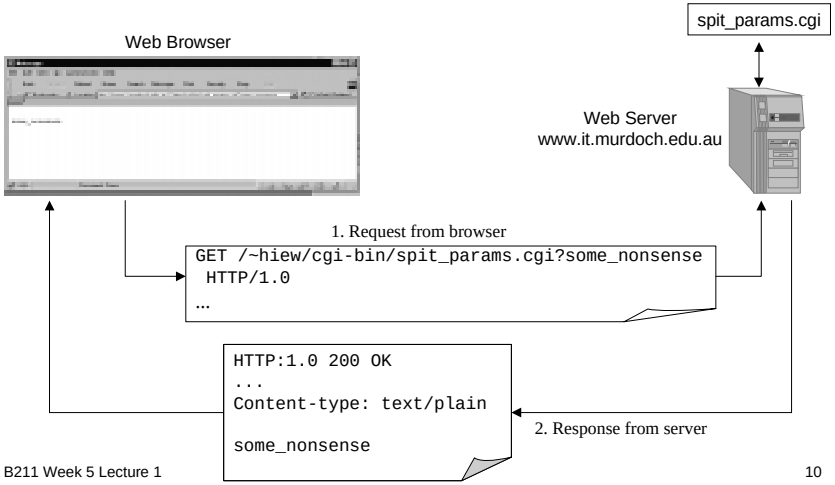
```
#!/usr/bin/perl -w

# ------------------------------------------------------------------------
#
# A CGI script to dump all the parameters passed to it back to the web
# client in plain text.
#
# Author: H.L. Hiew
# Date created: 15/07/2000
# Last modified: 10/08/2001
#
# ------------------------------------------------------------------------

#
# Put the submitted data into a variable $data
#
my $data;

if ($ENV{'REQUEST_METHOD'} eq "GET") {
        $data = $ENV{'QUERY_STRING'};
} else {
        $data = <STDIN>;
}

#
# Send the data back to the browser.
#
print <<END;
Content-type: text/plain\n\n
$data \n
END
```

spit_params.cgi

- Whatever parameters this script receives, it sends back to the browser.

Accessing the script from a browser

What actually happens:



Sending Data from an HTML form

send2script.html

```
<html>
  <body>
    <form method="GET"
     action="http://www.it.murdoch.edu.au/~hiew/cgi-bin/spit_params.cgi">
      Something to send: <input type="text" name="MyParameter">
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

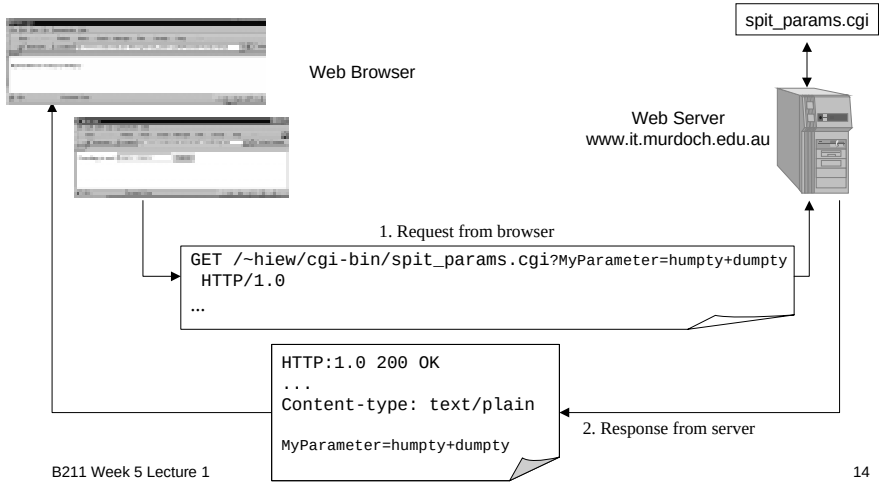
- Accessing spit_params.cgi from a form instead of typing it directly in the browser's URL box.

The form before submission

The result after pressing “Submit”

This is how the browser puts together the data when the method is “GET”.

What actually happens:



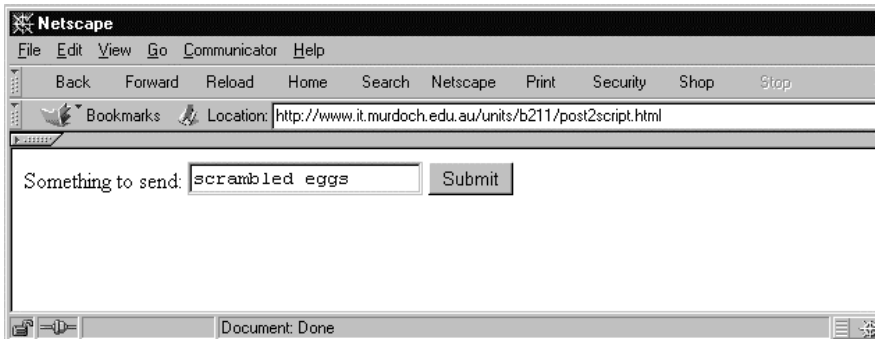
Using POST instead of GET

post2script.html

```
<html>
  <body>
    <form method=”POST"
     action="http://www.it.murdoch.edu.au/~hiew/cgi-bin/spit_params.cgi">
      Something to send: <input type="text" name="MyParameter">
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

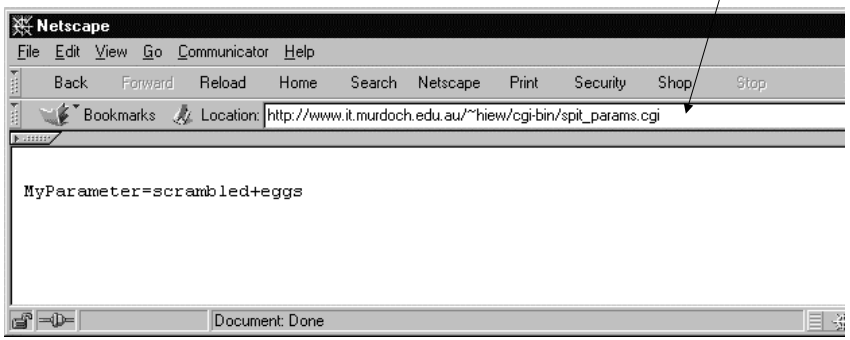
- Note the change from `method=“GET”` to `method=“POST”`

The form before submission

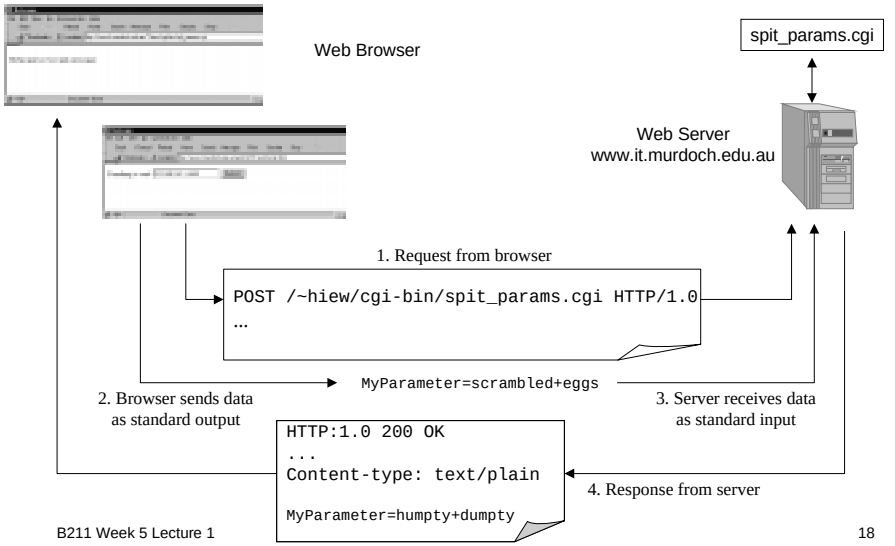


The result after pressing “Submit”

Note how the parameters are not part of the URL. Useful for passing data you do not want to appear on the screen.



A virtual “pipeline”:

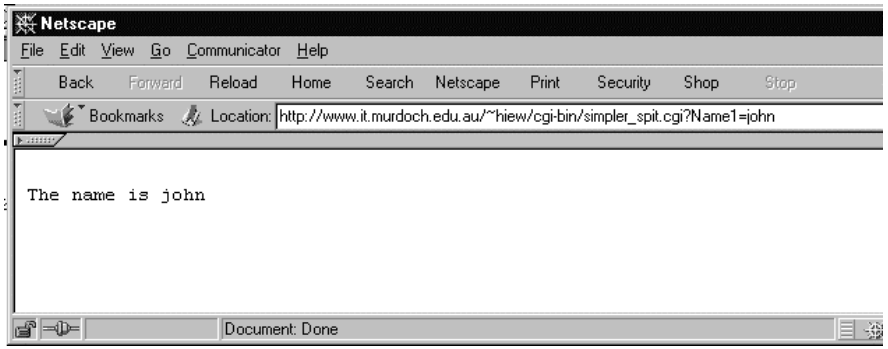


Using the CGI module in Perl

simpler_spit.cgi

```
#!/usr/bin/perl -w

# ------------------------------------------------------------------------
#
# A CGI script to receives the paramater named "Name1" and pass it back to
# the web client in plain text. Makes use of the CGI module.
#
# Author: H.L. Hiew
# Date created: 15/07/2000
# Last modified: 10/08/2001
#
# ------------------------------------------------------------------------

#
# Declare that we want to use the CGI module
#
use CGI qw(:standard) ;

#
# Put the submitted data into a variable and send the data back to the
# browser.
#
my $name = param("Name1");

print <<END;
Content-type: text/plain\n\n
The name is $name \n
END
```


Accessing the script from a browser



Returning results in HTML

html_results.cgi

```
#!/usr/bin/perl -w

# ------------------------------------------------------------------------
#
# A CGI script to receives the paramater named "Name1" and pass it back to
# the web client in HTML. Makes use of the CGI module.
#
# Author: H.L. Hiew
# Date created: 15/07/2000
# Last modified: 12/08/2001
#
# ------------------------------------------------------------------------

#
# Declare that we want to use the CGI module
#
use CGI qw(:standard) ;

#
# Put the submitted data into a variable
#
my $name = param("Name1");

print <<END;
Content-type: text/html\n\n

<html>
  <body>
    <h1>Result from html_results.cgi</h1>
    <p>The name is <i>$name</i>.</p>
  </body>
</html>
END
```


Accessing the script from a browser

Returning another form

- Because you can return the results in HTML, there is no reason why you can't return ***another form*** to the user to fill out, which in turn accesses ***the same or another script***.

Example

postname.html

```
<html>
  <body>
    <form method="POST"
     action="http://www.it.murdoch.edu.au/~hiew/cgi-bin/getname.cgi">
      Name: <input type="text" name="Name">
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

Example

getname.cgi

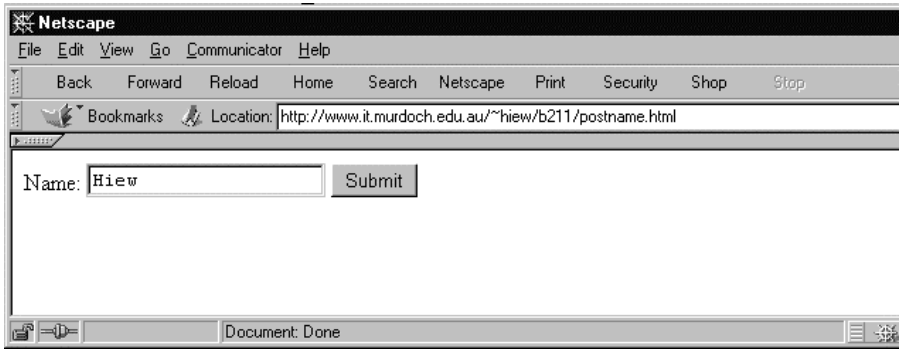
```
#!/usr/bin/perl -w

# ------------------------------------------------------------------------
#
# A CGI script to receive a name from the user, and return another HTML form
#
# Author: H.L. Hiew
# Date created: 15/07/2000
# Last modified: 15/08/2001
#
# ------------------------------------------------------------------------

use CGI qw(:standard) ;

#
# Put the submitted data into a variable, and print another HTML dorm to ask
# for the phone number.
#
my $name = param("Name");

print <<END;
Content-type: text/html\n\n

<html>
  <body>
    <p>Hi, $name. I need more information.</p>
    <form method="POST"
     action="getnamephone.cgi">
      Your Phone Number: <input type="text" Name="Phone">
      <input type="hidden" name="Name" value="$name">
      <input type="submit" value="Submit">
    </form>
  </body>
 </body>
</html>
END
```


Example

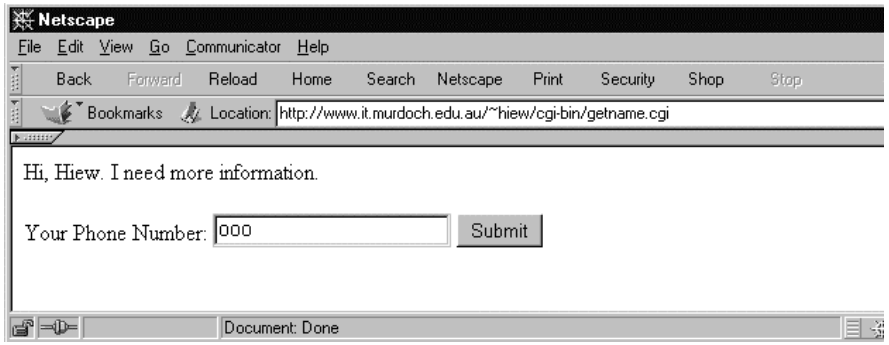
getnamephone.cgi

```
#!/usr/bin/perl -w

# ------------------------------------------------------------------------
#
# A CGI script to receive a name and a phone number from the user, and
# print that information back to the user.
#
# Author: H.L. Hiew
# Date created: 15/07/2000
# Last modified: 15/08/2001
#
# ------------------------------------------------------------------------

use CGI qw(:standard) ;

#
# Put the submitted data into variables
#
my $name = param("Name");
my $phone = param("Phone");

print <<END;
Content-type: text/html\n\n

<html>
  <body>
    <h1>Result from getnamephone.cgi</h1>
    <p>The phone number for $name is $phone.</p>
  </body>
</html>
END
```


The first form...



After submitting, we get another form...



After submitting again, we get the final result...

Why Server-side Scripting?

- HTML is static.
- JavaScript dynamic but only on the client-side
  - It can't get access to the resources on a server (eg. a database)
  - There are certain information which we cannot transfer down to the client to process (eg. passwords).
- Server-side technologies such as CGI allows us to give users access to services, but have the services processed by the server.

In the labs next week...

- In your lab exercises next week, you will be given instructions on how to get the above scripts to work in your own directory on red.
- What it essentially involves is:
  - Create script files above using a text editor (eg. Notepad) on your local machine.
  - FTP files onto the `public_html/cgi-bin` directory in your account on red.
  - Telnet/ssh to red and make the scripts executable.
  - Access the scripts using the appropriate URLs.

In the labs next week...

- You will also see an example of a search engine, where:
  - The passwords for users and the data for the searching are stored are on the server
  - The first form asks the user for a username and password.
  - The script only prints ***another form*** for the user to do the searching if the username/password is checked and are correct.
- Think about whether this is possible using only client-side technologies.