Putting a Program on the Web

Professor Don Colton, BYU Hawaii

November 26, 2008

Without the web, your program can only run on the computers where it is installed and no place else. But today's world connects almost all computers so your program can be shared and enjoyed by many people at the same time.

This introductory handout shows how to put a program on the web. We assume you already know how to write a program and test it locally.

1 Terms

local is a program running at the computer where you are sitting. It is not running on the web.

offline is the same as local.

online is a program that *is* running on the web.

web server is a computer that hosts web pages and web-based programs.

2 Sample: Counting Offline

Our first example is a program that counts from 1 to 500. The offline version of the program is fairly simple:

```
print "Don's Counting Program\n";
for ( $i = 1; $i <= 500; $i++ ) {
  print " $i" }
print "\nDone!\n";
```

You can test it offline on a Window computer by saving it to a file named, for example, `500.pl`, and double-clicking it. If `perl` is installed on your Windows computer, it should run.

If the program simply flashes on the screen and disappears, you might add the following line to make it stay on the screen so you can see the results. This line simply asks for input but does nothing with it. The fact the program is waiting for input will prevent it from closing the window where your results are displayed.

```
$wait = <STDIN>;
```

You can test offline by doing a `start` / `run` / `cmd` to open a command line window. In that window, type `cd desktop` and then `500.pl` to run your program.

3 Sample: Counting Online

By adding two lines, we can convert this program to run online.

```
#! /usr/local/bin/perl --
print "content-type: text/html\n\n";

print "Don's Counting Program\n";
for ( $i = 1; $i <= 500; $i++ ) {
  print " $i" }
print "\nDone!\n";
```

The first line tells the server how to run the program. It specifies where `perl` is to be found on the server.

The second line has the server tell the browser that the information it is about to receive is text, possibly including html markup.

4 Using a Web Server

You may be able to skip this section.

This section gives specific instructions using information about me. You will have to generalize from my specific examples to see how to do it in your own case.

You will need a web-hosting account. Such accounts are available (in 2008) for about $6 per month on a two-year contract. You may have received one from your instructor.

We assume the URL to your website is `http://doncolton.com/`

We assume your web server is running `linux`.

We assume you have shell access to your account by using something like `ssh`. It is usually possible to do most things without shell access, but shell access will make life easier.

We assume your account name is `doncolto`. (Account names are usually limited to eight characters.)

We assume your home directory is `/home/doncolto`. After logging in, you can type `cd` and then `pwd` to find what your home directory really is. We will call this `(home)`.

We assume the path to `perl` on the web server is `/usr/local/bin/perl`. You can type `which perl` to find the correct path to `perl` on your server.

We assume your web pages are stored in `(home)/public_html`

We assume your cgi scripts are stored in `(home)/public_html/cgi-bin`

We assume the URL to your cgi scripts is `http://doncolton.com/cgi-bin/`

Copy your program to a web server to this location:

`(home)/public_cgi/cgi-bin/500`

Set the permissions to 755.

```
chmod 755 (home)/public_cgi/cgi-bin/500
```

Test the program from the command line.

```
~/public_cgi/cgi-bin/500
```

It should start. You may need to key in a line of input or press enter to create a blank line of input. Make sure the program creates a valid web page.

You should now be able to run the program online by using the following URL.

`http://doncolton.com/cgi-bin/500`

It should display the numbers from 1 to 500 just as if it were running on your local computer.

See "What Could Go Wrong?" below if you are getting error like not found, forbidden, or internal server error.

5 Counting to N offline

Next we add some input to our offline program.

```
print "Don's Counting Program\n";
print "count how high? ";
$high = <STDIN>;
for ( $i = 1; $i <= $high; $i++ ) {
  print " $i" }
print "\nDone!\n";
```

We have added two lines (count how high, and stdin) and modified the loop to not always end at 500.

Test it to make sure it is working.

6 Counting to N online

Web programs do not stop in the middle to ask for input. Instead, we will run the program twice. The first time we will ask for input. The second time we will process it.

```
#! /usr/local/bin/perl --
print "content-type: text/html\n\n";

print "Don's Counting Program\n";
$input = <STDIN>;

if ( $input eq "\n" ) { # nothing entered

print "count how high? ";
print "<form method=post>";
print "<input name=high>";
exit }
```

```
$input =~ /high=(.*)/;
$high = $1;

for ( $i = 1; $i <= $high; $i++ ) {
  print " $i" }
print "\nDone!\n";
```

I will explain the new parts.

`$input = <STDIN>;`

This line accepts one line of input sent from the browser to your program on the server. (Note: you only get one line of input no matter how many pieces of data are being sent. See below for more details.)

`if ( $input eq "\n" ) { # nothing entered`

This line checks to see if the input was blank. If it was blank, we will ask the question. Otherwise we will generate the answer.

`print "<form method=post>";`

This line tells the browser that it can send us inputs.

`print "<input name=high>";`

This line tells the browser to make an input box on the web page and to name it `high`.

`exit`

This line tells the server to stop running your program. Everything you have printed will be sent to the browser. Nothing else will be done at this time.

`$input =~ /high=(.*)/;`

This line tells the server to look through the line of input it got. It is looking for the information sent by the browser. If the user keyed in the number `55`, the browser will send `high=55` to the server and the server will feed it to your program. Your program will look for `high=` followed by a number.

`$high = $1;`

This line takes the number found and copies it to a variable named `$high` for later use.

7 Summary

Putting a program on the web is fairly easy. We have looked at two cases.

In the first case there was no input. We got to the web by adding two lines to the program.

In the second case there *was* input. Our web program needed an `if` statement to select which thing to do. When no input was provided, the program had to ask for input and then stop. When input was provided, the program had to extract that input and then use it to run the program.

```
#! /usr/local/bin/perl --
print "content-type: text/html\n\n";
$input = <STDIN>;

(things to do every time)

if ( $input eq "\n" ) { # nothing entered
(things to do when input not provided)
exit }

(extract the inputs)

(things to do when input was provided)
```

8 Adding Numbers Offline

Finally, we will show how to get more than one input.

```
print "Don's Adding Program\n";
print "first number? ";
$first = <STDIN>;
print "second number? ";
$second = <STDIN>;
$answer = $first + $second;
print "The total is $answer\n";
```

9 Adding Numbers Online

```
#! /usr/local/bin/perl --
print "content-type: text/html\n\n";
$input = <STDIN>;
```

```
# things to do every time
print "Don's Adding Program\n";

if ( $input eq "\n" ) { # nothing entered
# things to do when input not provided
print "<form method=post>";
print "first number? ";
print "<input name=first>";
print "second number? ";
print "<input name=second>";
exit }

# extract the inputs
$input =~ /first=(.*)&second=(.*)/;
$first = $1;
$second = $2;

# things to do when input was provided
$answer = $first + $second;
print "The total is $answer\n";
```

The interesting part here is extracting the inputs.

Notice that each field is listed along with its own `=` sign and with its own number.

Notice that the first field is extracted into `$1` and the second field is extracted into `$2`.

Everything else should be fairly straightforward.

10 Simplifications

The extraction phase uses a technique called a regular expression. They are very powerful. Whole books have been written about regular expressions. All you really need to know at this point is that they can extract the information sent by the browser.

11 What's Next?

Using the techniques shown in this handout, try converting a few programs from local (offline) to online.

12 What Could Go Wrong?

There are several common errors that first-time programmers tend to encounter. If your page is not working, here are some things to check.

Not Found (404): If you get a message like this, make sure that your file is in your `~/public_html` directory. Sometimes people will mistakenly create another `public_html` directory inside their main `~/public_html` directory, or will put their files in their home directory instead. Use the `pwd` command (print working directory) to find out where you are editing and storing your file.

Forbidden: If you get a message like this, make sure your file permissions and directory permissions are set properly. Verify that the permissions on your program are set to `755`.

Internal Server Error (500): This means your program did not produce a viable web page. Usually it means your program crashed. It could also mean your program ran but created mal-formed outputs.

Run your program from the command line. See if it shows any error messages.

Verify that your `content-type` line is exactly right, with no extra spaces and no spelling errors (context vs content).

```
content-type: text/html\n\n
```

Verify that your permissions are set to `755`.