Perl Tutorial: Getting started with Perl CGI scripting

and how to avoid common problems beginners have when writing their first scripts

What is Perl/CGI? Perl is a simple programming language. It doesn't have to be used on the web, it can run locally on your computer, but it's popular for use on the web. When it's used on the web the programs are called Perl CGI, because CGI is the way that Perl talks to your web browser. Perl can be used to do things like rotate banners, generate text & HTML on the fly, set cookies, and provide shopping carts.

In theory it's pretty simple:

1. **Write your Perl program in a text editor,** and save it with a *.cgi* or *.pl* extension.
2. **Upload it to your web server.**
3. **Run it in one of three ways:**
   - Link to it. (e.g., *<a href=myscript.cgi>Click to run my program</a>*)
   - Embed it into your HTML file (e.g, *<p><!--#include virtual="myscript.cgi"-->*)
   - Use it as the action item of a form (e.g., *<form action=myscript.cgi>*)

But if you tried this already it probably didn't work, which is probably why you're here. That's good, we'll show you a whole bunch of ways you could have gone wrong, and then you'll be able to get your script working. When I started out I had a hard time finding good resources. If I had found a page as useful as this one I wouldn't have had to write it. Most of the Perl tutorials I found omitted crucial information, failed to give an overview or put things into context, had explanations that were either too long or not long enough, spread out the information over several pages instead of putting everything in one spot, and had all manner of annoying blinking advertisements, I wrote this tutorial to provide an alternative, and I wrote it to answer the question, "What kind of resource do I wish I had when I was starting out?" So I think you're in good hands.

By the way, I use the terms "program" and "script" in this tutorial interchangeably. They're the same thing.

And now for the good stuff.

Steps to creating a successful Perl CGI script

1. Get some info from your webhost. Your webhost is the company that has your website on their servers. Having a webhost which properly supports Perl is half the battle. We use Dreamhost, and if you use them then everything on this page should work well for you, and you can skip this step and go to the next one. If your host isn't Dreamhost then you'll need to ask them some questions:

- Do you support Perl? (If not, stop here and get another host that does, like Dreamhost.)
- Can I use the standard shebang line? `#!/usr/bin/perl`
- Can I use the **.cgi** extension or do I have to use **.pl**?
- Can my files go anywhere, or do they have to go in a **cgi-bin** directory?
- What permissions do I need to set for the script and for the directory it's in?
- Where are my server error logs located?

We'll use the answers to these questions in the steps below.

2. Write your script with a text editor. Many Windows text editors put carriage returns at the ends of lines which can cause Perl scripts to fail. *Any* of these will solve that problem:

- **Use Unix.** Use a text editor in the Unix shell such as `pico`
- **Use a Mac.** On Mac OS X, use **TextEdit** (and choose Format > Make Plain Text) and make sure to save your file without the ".txt" on the end. Or, use the built-in editor in the Transmit FTP/SFTP file transfer software.
- **Windows:** Neither NotePad nor WordPad saves in the proper format, even if you choose Text. (It's not the right kind of Text.) You'll need to either log into the Unix shell and use an editor like pico, or find some other text editor for Windows that saves in the right format, such as TextPad.

3. Your first script. Type this in (or copy & paste it) into a new file:

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

print "Content-type: text/html\n\n";
print "Hello, world";
```

4. Save your file with a .cgi extension (e.g., "hello.cgi"). Some webhosts might require a **.pl** extension, check with them to find out. With Dreamhost you can use either .cgi or .pl.

5. Upload your file. Check with your webhost to see if you have to put it in a special place. Some hosts require that it go in the **cgi-bin** directory. If your host is Dreamhost you can put it anywhere.

6. Use ASCII mode for the upload. If you upload in binary mode it won't work.

7. Set permissions. You need to set permissions, which is a fancy way of telling the server that you're authorizing the program to run there. You do this with the Unix command line. (See more about the Unix command line below if it's unfamiliar to you.) Check with your webhost to see what permissions they require. At Dreamhost you set permissions with **chmod 755** for both your file and the directory it's in. If you rename the file you need to set its permissions again. If you delete the file from the server and then upload a fresh copy you need to set its permissions again. If you move it to another directory, you have to set the permissions at that new directory.

9. Run the script. There are four different ways to do this:

- **As a url** (link to the file, e.g., `http://mydomain.com/hello.cgi`, or type it into a browser window)
- **Embedded anywhere in a web page:** `<!--#include virtual="hello.cgi"-->`
  See note below.
- **As the action for a FORM:** `<FORM ACTION="hello.cgi" METHOD=POST>`
- **At the Unix command line:** `perl hello.cgi`

Notes:

If your script is in a cgi-bin directory, don't forget to include that when typing the url.

If this didn't work then see our troubleshooting section.

Embedding the reference in a web page:

The whole page might look like this:

```
<HTML>
<BODY>
<P>Here's the output from my program:
<!--#include virtual="hello.cgi"-->
</P>
</BODY>
</HTML>
```

To get the server to run the CGI from your HTML page you also have to do *one* of the following:

- Title the page with a **`.shtml`** extension (e.g., hello.shtml), OR
- Add this line to the **.htaccess** file that goes at the top level of your website (same level as your index.html file):
  - **`AddHandler server-parsed .html`**

  If you don't already have an .htaccess file, then create one and have the AddHandler bit be the only line.

The Unix Command Line tutorial

and how to set file permissions with it

You'll use the Unix command line to set the permissions for your file, and possibly to test it for syntax errors. Check with your webhost to see what permissions they require. At Dreamhost you need to **chmod 755** both your file and the directory it resides in.

If you're unfamiliar with the Unix command line, keep reading.

Where to get it

Lucky for you, Unix command line interfaces come preinstalled with both Macs and Windows. On Mac OS X just open the Terminal application that's in Applications/Utilities on your hard drive. With Windows 98 go to Start > Run > `c:\windows\telnet` I'm not sure where it is in other versions of Windows, you may need to use the Find command.

Security

There are two ways to connect to your webhosting account via the Unix command line: Telnet and SSH. Of these SSH is preferred because it's secure -- your password is encrypted so no one can pick it up as it makes its way through the Internet. The Mac OS X Terminal supports both Telnet and SSH. But Windows Telnet only does Telnet, not SSH. You can either use Telnet anyway, or look for SSH software for Windows. Go to download.com and search for "ssh".

Logging in

You'll be using the same username & domain name combo that you use to upload your web files to your server. Do NOT include the www when typing your domain name.

In the Mac OS X Terminal, type:

```
ssh user@domain.com
```

Then you're prompted for your password.

In Windows Telnet:

Go to Connect > Remote System. For host, put in your domain name WITHOUT the www. Then you'll be prompted to enter your username and password.

Unix commands

Hooray, you've logged in. So now what can you do? Here are some basic commands

`ls`	Lists the contents of the directory
`cd` *`directory name`*	Change the directory. For example, you could do cd domain.com or cd domain.com/test When you first log in at Dreamhost (and many other hosts) you need to go down one level to get to the public HTML files, just like you do with FTP. This means doing a cd domain.com as your very first command.

	You need to be in the directory of the files you're working with, unless you want to have to do a lot of extra typing To back up one level, type **cd ..**
`pico filename.cgi`	A text editor, lets you view and/or edit a file. There are lots of commands within pico that I'm not going to cover, except ^X (Control-X) to exit pico.
`chmod filename or directory name`	Sets file permissions. More on this below.
`perl -w filename.cgi`	Runs a perl script
`(up arrow)`	Recalls previously-typed commands so you don't have to type them again.

Set permissions

You'll use this command to set the permissions for your file. That's just a fancy way of saying that it makes the file authorized to run on your server.

chmod 755 is specific to Dreamhost. Other servers may use something different. And for Dreamhost, even if the instructions that came with a preinstalled script tell you to use something besides 755, ignore it and use 755 anyway.

For example, at Dreamhost, if your file is called **myprogram.cgi** and is in a directory called **scripts** then after logging in you'd type:

```
cd mydomain.com
chmod 755 scripts
cd scripts
chmod 755 myprogram.cgi
```

Script Basics

Things to know about every Perl CGI script you write

The shebang line

The first line of a script is called the shebang line. It should look like this:

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

usr/bin/perl tells the system where the Perl interpreter is located.

The -w switch tells the interpreter to turn on Warnings about possible problems with your code. This will help you with your debugging.

Special print command

In plain Perl, all you have to do to print something is use the print command. But with Perl CGI for the web you have to include this line before your first print command:

```
print "Content-type: text/html\n\n";
```

This is called an http header and tells the browser what kind of content it's about to get (in this case text/html, as opposed to, say, a cookie). **If you don't include this then you'll get an Internal Server Error when you try to run it.**

The \n's are carriage returns. Don't worry, they won't add extra carriage returns to your outputted web page, they're just part of the http header (and necessary).

Note that you print the header *once*, before your very first print command (not before *every* print command).

Here's an alternate way to print:

```
use CGI;
$query = new CGI;
print $query->h3('This is a headline.');
print $query->p('This is body text.');
```

Must include output

Your script has to actually print some output or you'll get an Internal Server Error when you try to run it in a browser. This script would fail for that reason:

```
#!/usr/bin/perl -w
$variable="value";
```

You wouldn't get any errors when running it from the Unix command line, but it wouldn't work in a browser. The error in your error log would say, "Premature end of script headers."

But this script would work:

```
#!/usr/bin/perl -w
$variable="value";
print "Content-type: text/html\n\n";
                   print $variable;
```

Important! This seems so simple you'll be tempted to blow it off, because off course your program will have output, right? But here's how you'll run into it: You'll be getting some error because of something else, so you'll start stripping away parts of your program to test to see where the error is, to see if you've removed the offending code. Well you might remove the offending code *and* your only print statement, which fixes one problem and creates another. And don't count on a print statement that depends on an IF command. For safety's sake, put this line at the end of your code and keep it there until your development is done:

```
print "Content-type: text/html\n\n";
```

TROUBLESHOOTING -- Those damn "Internal Server Errors"

Errors are a fact of life

It seems like everything causes an error! I'm the most novice of novice programmers. And I had a hell of a time getting my Perl scripts to run in a browser without giving Internal Server Errors. I'd copy some simple example I found on the net, but it would never work. I'd write the simplest of programs, such as:

```
#!/usr/bin/perl

$variable="value";
```

and I'd *still* get an Internal Server Error! After a lot of effort finding out what causes the errors and how to correct them, I wrote this tutorial to share what I learned so your learning curve won't be as painful as mine was. That's why I wrote this article. Below is the summary of all the things I found that cause Internal Server Errors.

Summary of things that cause Internal Server Errors

- **Directory or file not set to the proper permissions.** Check with your webhost about what they require. At Dreamhost, it's **chmod 755** for both the file and the directory it's in. Note that *you may have to chmod the file *again* after you've already set it.* For example:
    - If you rename the file, you may have to chmod it again.
    - If you delete the file from the server and then re-upload it from your hard disk, you may have to chmod it again.
    - If you tried to run the file in a web browser and got an Internal Server Error because you forgot to print the header (see below), you may have to chmod it again.
- **File in wrong location.** Some webhosts require you to put the file in a special place, such as a cgi-bin directory. At Dreamhost, it doesn't matter where the file goes, you can put it anywhere.
- **File has the wrong extension.** On most servers the filename should end with **.cgi**. Check with your webhost to find out what it is on your server.
- **File not in Unix text format.** Most Mac & PC text editors put carriage returns at the ends of lines, which can cause Perl scripts to fail. Either use a Unix/Linux text editor, or find a Mac or PC text editor that saves in Unix format (like BBEdit).

- **Missing print header.** Before you print something to the browser you have to use this line first: print "Content-type: text/html\n\n";
- **Program doesn't do any output.** A script that just messes with variables and doesn't actually print anything can give an error.
- **File transferred to server in binary mode instead of ASCII mode.** Set your FTP or SSH software to ASCII mode. If you're using the Unix command line to transfer, just type `ascii` and hit Enter.
- **Missing or incorrect shebang line.** The first line of every script should be: `#!/usr/bin/perl -w` The -w is optional but it's a good idea (more on this above). The pathname may be different on your system. Type `which perl` in the Unix command line to find the pathname to use for your system. You will always start it with #! no matter what you get from `which perl`. Also, make sure you typed #! instead of !#, a common mistake.
- **Capitalized commands.** At least on the server I'm using "IF" breaks my code while "if" works fine.
- **Any syntax error,** especially missing a semicolon at the end of a line.
- **Invisible garbage characters.** If you copy & paste code from a web page you might be copying invisible characters that kill your script (usually at the end of a line). Replace the existing returns & tabs by selecting them and typing fresh returns and tabs over them.

How to see the *actual* error, instead of "Internal Server Error"

Put this code at the top of your script, and then when there's an error your browser will report the *actual* error:

```
use CGI::Carp qw/fatalsToBrowser/;
```

This will report syntax and logic errors, but not "bad file" errors. If you still get an Internal Server Error, then likely your file is not chmod 755, or it's missing the print header.

What to do when you have an Internal Server Error

Don't panic.

First of all, don't panic. When I first learned Perl CGI it seemed that *all* I got was Internal Server Errors and I felt like giving up. But I stuck with it and now I've been able to code all manner of useful things like rotating adverts and custom shopping carts. Stick with it, Perl works, you'll just have to do some tweaking. This troubleshooting guide will help.

Run the file from the Unix command line

If you get an Internal Server Error try running the file from the Unix command line, e.g.: `perl -w hello.cgi`

First of all, if there's a problem the Unix command line might give you a more specific error message that can help you track down the problem. (My favorite is "possible missing semicolon".) If it runs just fine then that's good too, because it tells you that your problem is one of the following:

- **File not in Unix format.** Remember to save the file in Unix text format. You may need to use a special Unix text editor.
- **File permissions not set correctly** (Did you rename, move, or recopy the file? Did you never set permissions in the first place?)
- **Missing "print" header.** Don't forget the print "Content-type: text/html\n\n"; we covered above.
- **No output.** Remember that with CGI your script needs to actually print something. Just setting variables alone won't cut it.

In the command line, an error saying that "a variable might have only been used once" isn't serious and that alone won't cause an Internal Server Error when the script is run in the browser. But it could mean that you misspelled a variable name somewhere, and your script may not do what you want it to do.

Check the server log

If you get an Internal Server Error in the browser but the script runs fine from the Unix command line, and your permissions are set correctly, and you can't find any problems with your output/print code, then you can see if there's a more detailed error listed in your server's error logfile. On Dreamhost servers the error log is at **logs/domain.com/http/error.log**. Note that the logs directory is at the very top level, above the directory for your html files. From the Unix command line, you can view the last error with the `tail` command. At Dreamhost the full command would be `tail -f /home/username/logs/domain.com/http/error.log`, substituting your own domain name for domain.com of course.

Get rid of invisible garbage characters

If you copy & paste code from a web page you might be copying invisible characters that kill your script (usually at the end of a line). Replace the existing returns & tabs by selecting them and typing fresh returns and tabs over them.

Rebuild your script from scratch a few lines at a time

If you've done everything above and you're still getting errors then you're probably pretty frustrated at this point. I know, I've been there.

What you can do is to start with a basic script and build up from there. Take your existing script and back it up somewhere. Then start from scratch with a basic Perl program:

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

print "Content-type: text/html\n\n";
print "Hello, world";
```

Upload it and run it. Just running something that actually works can give you some satisfaction and help restore your faith in Perl. (And if *this* doesn't work, your problem almost certainly one of the ones listed above under "Run the file from the Unix command line". Run the file from the command line to make sure.)

After getting Hello World to work, take your broken script and copy & paste the first few lines to the end of your working "Hello world" script. Upload it and run it. Does it work? Great! Keep adding more lines back in until it breaks. Did it not work? Aha! Then you know that whatever you pasted in is the problem.

It's important when you do this that you yank *all* of your code out except for the three lines above, and that you do include those three lines. If you yank only part of your code out, you might yank out the part that has the problem, but now that your code is incomplete it'll generate an Internal Server Error for a *different* reason, but you won't know that. To you it will look like you didn't extract the bad code, when in fact you did. Since you mistakenly think the bad code still remains then it'll take you hours to figure out that you simultaneously fixed and re-broke your script at the same time.

Also remember that when you yank out any or all of your code make sure you don't remove your only print command! Otherwise you'll get an Internal Server Error because your script doesn't print, and just like in the example above, that won't be obvious -- you will have fixed and re-broken your script at the same time but that won't be obvious and then you'll be worse off than before.

Basic Perl Syntax tutorial

This section assumes you have some familiarity with some other programming language. If you don't please see the references at the end of this page.

Shebang line

Unlike Java and C++, you don't have to put a whole bunch of crazy header commands in your script, nor do you have to make sure that something *inside* the script matches the filename. Just put this as your first line in your script and you'll be fine:

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

You don't have to close the file with anything special, either. Nor do you have to define classes or functions or procedures. (You can define functions if you want, but you certainly don't have to.) Just throw in the shebang line, add any more commands you want, and that's it.

Lines end with semicolons

Except for the shebang line, every command must have a semicolon at the very end.

It's fine for a command to span multiple lines, though. For example:

```
$thepoem = "
   My poetry
   is so beautiful
   it can annihilate
   penguins";
```

Assigning Variables

You don't have to declare variables and you don't have to specify their type. The only thing special is that you have to put a **$** before the variable name.

```
$theText = "boo boo puppy";
$theTotal = 567;
```

Comments

The **#** sign indicates a comment, except in the shebang line. For example:

```
$theText = "boo boo puppy";  # this is my string variable
$theTotal = 567;             # this is my number variable
```

Numbers

Numbers work the way they do in other languages

```
$apples = 5;
$bananas= 6;
$total = $apples + $bananas;  # add apples & bananas together
$total = $total * 1.0825;     # add sales tax
$counter++;                   # add 1 to the counter
```

Turning a string into an integer is easy:

```
$size = int(size);
```

Generating random numbers is also easy:

```
 $x = int(rand(5));  # returns an integer between 0 and 4, inclusive
```

Strings

Strings are enclosed with either single or double quotes. Double quotes tell Perl to include variables that have already been defined.

```
$flavor = 'chocolate';
$food   = "$flavor cookies";
print "I like $food.";  #  Prints "I like chocolate cookies."
```

You can also combine strings with a period, though combining them as shown above is usually easier.

Use single quotes when you're not including variables, because that saves processor time. Yeah, the difference is negligible, but it's good practice.

You can escape quote marks the regular way:

```
$variable = 'Don\'t touch O\'Hara's \'snickerdoodle\'.';
```

But a better way is to use some other character to enclose your variable, by prefacing it with a q:

```
$variable = q[Don't touch O'Hara's 'snickerdoodle'.];
```

In fact you can use any character you want, like q#...#, or q!...!.

If your'e including other variables, use two qq instead of q:

```
print qq[I like $food."];
```

Some more handy string functions:

```
$theLength = length($variable); # number of characters in the variable
$chop($variable);  # removes last character
$chomp($variable); # removes last character only if it's a Return
```

Arrays

Arrays are specified with the @ sign.

```
@flavors = ('chocolate', 'strawberry', 'alarm clock');
```

And here's a shortcut to define arrays where each element is a single word:

```
@flavors = qw(chocolate strawberry vanilla);
```

Arrays are zero-based. (The first element is 0, not 1.) You refer to them just like in other languages:

```
print $flavors[0]; # prints 'chocolate'
```

$# returns the number of items in the array

```
print $#flavors; # Returns 2; the length is zero-based, too
```

Add and remove items to/from an array:

```
shift(@array)     # removes the first item
pop(@array)       # remove the last item
unshift(@array,newelement)  # adds to the beginning
push(@array,newelement1,newelement2)   # adds to the end
```

For loops

You can use **for** loops the same way you do in other languages:

```
for ($counter=0; $counter<10; counter++) {
    [commands go here];
}
```

But Perl has an easier way to do the same thing::

```
for $counter (0..10) {
    [commands go here];
}
```

You can process every item in an array the same way:

```
for $item (@myArray) {
    $item = "funky $item";
}
```

This modifies the item in the array itself.

I'm sorry, but I can't provide personal assistance with Perl. I have thousands of messages in my In Box as it is. I hope this tutorial helps you get off to a good start, though.

More resources:

- Beginner's Introduction to Perl
- Perl Reference Guide
- Robert's Perl tutorial
- The Perl you need to know

Fan Mail

Your website was very helpful. I was seeing so many SERVER ERRORs that I was ready to use my wood maul as a debugging tool. -- *Matt Alexander, June 2006*

Last Update: June 2006