Perl is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, web development, network programming, GUI development, and more.

What is Perl?

- Perl is a stable, cross platform programming language.
- Though Perl is not officially an acronym but few people used it as **Practical Extraction and Report Language**.
- It is used for mission critical projects in the public and private sectors.
- Perl is an *Open Source* software, licensed under its *Artistic License*, or the *GNU General Public License (GPL)*.
- Perl was created by Larry Wall.
- Perl 1.0 was released to usenet's alt.comp.sources in 1987.
- At the time of writing this tutorial, the latest version of perl was 5.16.2.
- Perl is listed in the *Oxford English Dictionary*.

PC Magazine announced Perl as the finalist for its 1998 Technical Excellence Award in the Development Tool category.

Perl Features

- Perl takes the best features from other languages, such as C, awk, sed, sh, and BASIC, among others.
- Perls database integration interface DBI supports third-party databases including Oracle, Sybase, Postgres, MySQL and others.
- Perl works with HTML, XML, and other mark-up languages.
- Perl supports Unicode.
- Perl is Y2K compliant.
- Perl supports both procedural and object-oriented programming.
- Perl interfaces with external C/C++ libraries through XS or SWIG.
- Perl is extensible. There are over 20,000 third party modules available from the Comprehensive Perl Archive Network (CPAN).
- The Perl interpreter can be embedded into other systems.

Perl and the Web

- Perl used to be the most popular web programming language due to its text manipulation capabilities and rapid development cycle.
- Perl is widely known as "the duct-tape of the Internet".
- Perl can handle encrypted Web data, including e-commerce transactions.
- Perl can be embedded into web servers to speed up processing by as much as 2000%.
- Perl's mod_perl allows the Apache web server to embed a Perl interpreter.
- Perl's DBI package makes web-database integration easy.

Perl is Interpreted

Perl is an interpreted language, which means that your code can be run as is, without a compilation stage that creates a non portable executable program.

Traditional compilers convert programs into machine language. When you run a Perl program, it's first compiled into a byte code, which is then converted (as the program runs) into machine instructions. So it is not quite the same as shells, or Tcl, which are **strictly** interpreted without an intermediate representation.

It is also not like most versions of C or C++, which are compiled directly into a machine dependent format. It is somewhere in between, along with *Python* and *awk* and Emacs .elc files.

Perl - Environment

Before we start writing our Perl programs, let's understand how to setup our Perl environment. Perl is available on a wide variety of platforms −

- Unix (Solaris, Linux, FreeBSD, AIX, HP/UX, SunOS, IRIX etc.)
- Win 9x/NT/2000/
- WinCE
- Macintosh (PPC, 68K)
- Solaris (x86, SPARC)
- OpenVMS
- Alpha (7.2 and later)
- Symbian
- Debian GNU/kFreeBSD
- MirOS BSD
- And many more...

This is more likely that your system will have perl installed on it. Just try giving the following command at the $ prompt −

$perl -v

If you have perl installed on your machine, then you will get a message something as follows −

This is perl 5, version 16, subversion 2 (v5.16.2) built for i686-linux

Copyright 1987-2012, Larry Wall

Perl may be copied only under the terms of either the Artistic License or the GNU General Public License, which may be found in the Perl 5 source kit.

Complete documentation for Perl, including FAQ lists, should be found on this system using "man perl" or "perldoc perl". If you have access to the Internet, point your browser at http://www.perl.org/, the Perl Home Page.

If you do not have perl already installed, then proceed to the next section.

Getting Perl Installation

The most up-to-date and current source code, binaries, documentation, news, etc. are available at the official website of Perl.

Perl Official Website − https://www.perl.org/

You can download Perl documentation from the following site.

Perl Documentation Website − https://perldoc.perl.org

Install Perl

Perl distribution is available for a wide variety of platforms. You need to download only the binary code applicable for your platform and install Perl.

If the binary code for your platform is not available, you need a C compiler to compile the source code manually. Compiling the source code offers more flexibility in terms of choice of features that you require in your installation.

Here is a quick overview of installing Perl on various platforms.

Unix and Linux Installation

Here are the simple steps to install Perl on Unix/Linux machine.

- Open a Web browser and go to https://www.perl.org/get.html.
- Follow the link to download zipped source code available for Unix/Linux.
- Download **perl-5.x.y.tar.gz** file and issue the following commands at $ prompt.

```
$tar -xzf perl-5.x.y.tar.gz
$cd perl-5.x.y
$./Configure -de
$make
$make test
$make install
```

NOTE − Here $ is a Unix prompt where you type your command, so make sure you are not typing $ while typing the above mentioned commands.

This will install Perl in a standard location */usr/local/bin* and its libraries are installed in */usr/local/lib/perlXX*, where XX is the version of Perl that you are using.

It will take a while to compile the source code after issuing the **make** command. Once installation is done, you can issue **perl -v** command at $ prompt to check perl installation. If everything is fine, then it will display message like we have shown above.

Windows Installation

Here are the steps to install Perl on Windows machine.

- Follow the link for the Strawberry Perl installation on Windows http://strawberryperl.com
- Download either 32bit or 64bit version of installation.

- Run the downloaded file by double-clicking it in Windows Explorer. This brings up the Perl install wizard, which is really easy to use. Just accept the default settings, wait until the installation is finished, and you're ready to roll!

Macintosh Installation

In order to build your own version of Perl, you will need 'make', which is part of the Apples developer tools usually supplied with Mac OS install DVDs. You do not need the latest version of Xcode (which is now charged for) in order to install make.

Here are the simple steps to install Perl on Mac OS X machine.

- Open a Web browser and go to https://www.perl.org/get.html.
- Follow the link to download zipped source code available for Mac OS X.
- Download **perl-5.x.y.tar.gz** file and issue the following commands at $ prompt.

```
$tar -xzf perl-5.x.y.tar.gz
$cd perl-5.x.y
$./Configure -de
$make
$make test
$make install
```

This will install Perl in a standard location */usr/local/bin* and its libraries are installed in */usr/local/lib/perlXX*, where XX is the version of Perl that you are using.

Running Perl

The following are the different ways to start Perl.

Interactive Interpreter

You can enter **perl** and start coding right away in the interactive interpreter by starting it from the command line. You can do this from Unix, DOS, or any other system, which provides you a command-line interpreter or shell window.

```
$perl  -e <perl code>          # Unix/Linux

or

C:>perl -e <perl code>         # Windows/DOS
```

Here is the list of all the available command line options −

Sr.No.	Option & Description
1	**-d[:debugger]** Runs program under debugger
2	**-Idirectory**

	Specifies @INC/#include directory
3	**-T** Enables tainting checks
4	**-t** Enables tainting warnings
5	**-U** Allows unsafe operations
6	**-w** Enables many useful warnings
7	**-W** Enables all warnings
8	**-X** Disables all warnings
9	**-e program** Runs Perl script sent in as program
10	**file** Runs Perl script from a given file

Script from the Command-line

A Perl script is a text file, which keeps perl code in it and it can be executed at the command line by invoking the interpreter on your application, as in the following −

```
$perl  script.pl        # Unix/Linux

or

C:>perl script.pl       # Windows/DOS
```

Integrated Development Environment

You can run Perl from a graphical user interface (GUI) environment as well. All you need is a GUI application on your system that supports Perl. You can

download Padre, the Perl IDE. You can also use Eclipse Plugin EPIC - Perl Editor and IDE for Eclipse if you are familiar with Eclipse.

Before proceeding to the next chapter, make sure your environment is properly setup and working perfectly fine. If you are not able to setup the environment properly then you can take help from your system admininstrator.

All the examples given in subsequent chapters have been executed with v5.16.2 version available on the CentOS flavor of Linux.

Perl - Syntax Overview

Previous Page

Next Page

COMPLETE PERL Programming
11 Lectures 1.5 hours

Harshit Srivastava

Perl For Beginners: Learn A To Z Of Perl Scripting Hands-On
30 Lectures 6 hours

TELCOMA Global

Perl borrows syntax and concepts from many languages: awk, sed, C, Bourne Shell, Smalltalk, Lisp and even English. However, there are some definite differences between the languages. This chapter is designd to quickly get you up to speed on the syntax that is expected in Perl.

A Perl program consists of a sequence of declarations and statements, which run from the top to the bottom. Loops, subroutines, and other control structures allow you to jump around within the code. Every simple statement must end with a semicolon (;).

Perl is a free-form language: you can format and indent it however you like. Whitespace serves mostly to separate tokens, unlike languages like Python where it is an important part of the syntax, or Fortran where it is immaterial.

First Perl Program

Interactive Mode Programming

You can use Perl interpreter with **-e** option at command line, which lets you execute Perl statements from the command line. Let's try something at $ prompt as follows −

```
$perl -e 'print "Hello World\n"'
```

This execution will produce the following result −

```
Hello, world
```

Script Mode Programming

Assuming you are already on $ prompt, let's open a text file hello.pl using vi or vim editor and put the following lines inside your file.

Live Demo

```
#!/usr/bin/perl

# This will print "Hello, World"
print "Hello, world\n";
```

Here **/usr/bin/perl** is actual the perl interpreter binary. Before you execute your script, be sure to change the mode of the script file and give execution priviledge, generally a setting of 0755 works perfectly and finally you execute the above script as follows −

```
$chmod 0755 hello.pl
$./hello.pl
```

This execution will produce the following result −

```
Hello, world
```

You can use parentheses for functions arguments or omit them according to your personal taste. They are only required occasionally to clarify the issues of precedence. Following two statements produce the same result.

```
print("Hello, world\n");
print "Hello, world\n";
```

Perl File Extension

A Perl script can be created inside of any normal simple-text editor program. There are several programs available for every type of platform. There are many programs designd for programmers available for download on the web.

As a Perl convention, a Perl file must be saved with a .pl or .PL file extension in order to be recognized as a functioning Perl script. File names can contain numbers, symbols, and letters but must not contain a space. Use an underscore (_) in places of spaces.

Comments in Perl

Comments in any programming language are friends of developers. Comments can be used to make program user friendly and they are simply skipped by the interpreter without impacting the code functionality. For example, in the above program, a line starting with hash **#** is a comment.

Simply saying comments in Perl start with a hash symbol and run to the end of the line −

```
# This is a comment in perl
```

Lines starting with = are interpreted as the start of a section of embedded documentation (pod), and all subsequent lines until the next =cut are ignored by the compiler. Following is the example −

Live Demo

```
#!/usr/bin/perl

# This is a single line comment
print "Hello, world\n";

=begin comment
This is all part of multiline comment.
You can use as many lines as you like
These comments will be ignored by the
compiler until the next =cut is encountered.
=cut
```

This will produce the following result −

```
Hello, world
```

Single and Double Quotes in Perl

You can use double quotes or single quotes around literal strings as follows −

Live Demo

```
#!/usr/bin/perl

print "Hello, world\n";
print 'Hello, world\n';
```

This will produce the following result −

```
Hello, world
Hello, world\n$
```

There is an important difference in single and double quotes. Only double quotes **interpolate** variables and special characters such as newlines \n, whereas single quote does not interpolate any variable or special character. Check below example where we are using $a as a variable to store a value and later printing that value −

Live Demo

```
#!/usr/bin/perl

$a = 10;
print "Value of a = $a\n";
print 'Value of a = $a\n';
```

This will produce the following result −

```
Value of a = 10
Value of a = $a\n$
```

Perl Identifiers

A Perl identifier is a name used to identify a variable, function, class, module, or other object. A Perl variable name starts with either $, @ or % followed by zero or more letters, underscores, and digits (0 to 9).

Perl does not allow punctuation characters such as @, $, and % within identifiers. Perl is a **case sensitive** programming language. Thus **$Manpower** and **$manpower** are two different identifiers in Perl.

Perl - Data Types

Perl is a loosely typed language and there is no need to specify a type for your data while using in your program. The Perl interpreter will choose the type based on the context of the data itself.

Perl has three basic data types: scalars, arrays of scalars, and hashes of scalars, also known as associative arrays. Here is a little detail about these data types.

Sr.No.	Types & Description
1	**Scalar** Scalars are simple variables. They are preceded by a dollar sign ($). A scalar is either a number, a string, or a reference. A reference is actually an address of a variable, which we will see in the upcoming chapters.
2	**Arrays** Arrays are ordered lists of scalars that you access with a numeric index, which starts with 0. They are preceded by an "at" sign (@).
3	**Hashes** Hashes are unordered sets of key/value pairs that you access using the keys as subscripts. They are preceded by a percent sign (%).

Perl - Variables

Variables are the reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or strings in these variables.

We have learnt that Perl has the following three basic data types −

- Scalars
- Arrays
- Hashes

Accordingly, we are going to use three types of variables in Perl. A **scalar** variable will precede by a dollar sign ($) and it can store either a number, a string, or a reference. An **array** variable will precede by sign @ and it will store ordered lists of scalars. Finaly, the **Hash** variable will precede by sign % and will be used to store sets of key/value pairs.

Perl maintains every variable type in a separate namespace. So you can, without fear of conflict, use the same name for a scalar variable, an array, or a hash. This means that $foo and @foo are two different variables.

Creating Variables

Perl variables do not have to be explicitly declared to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

Keep a note that this is mandatory to declare a variable before we use it if we use **use strict** statement in our program.

The operand to the left of the = operator is the name of the variable, and the operand to the right of the = operator is the value stored in the variable. For example −

```
$age = 25;            # An integer assignment
$name = "John Paul";   # A string
$salary = 1445.50;     # A floating point
```

Here 25, "John Paul" and 1445.50 are the values assigned to *$age*, *$name* and *$salary* variables, respectively. Shortly we will see how we can assign values to arrays and hashes.

Scalar Variables

A scalar is a single unit of data. That data might be an integer number, floating point, a character, a string, a paragraph, or an entire web page. Simply saying it could be anything, but only a single thing.

Here is a simple example of using scalar variables −

Live Demo

```
#!/usr/bin/perl

$age = 25;            # An integer assignment
$name = "John Paul";   # A string
```

```
$salary = 1445.50;    # A floating point

print "Age = $age\n";
print "Name = $name\n";
print "Salary = $salary\n";
```

This will produce the following result −

```
Age = 25
Name = John Paul
Salary = 1445.5
```

Array Variables

An array is a variable that stores an ordered list of scalar values. Array variables are preceded by an "at" (@) sign. To refer to a single element of an array, you will use the dollar sign ($) with the variable name followed by the index of the element in square brackets.

Here is a simple example of using array variables −

Live Demo

```
#!/usr/bin/perl

@ages = (25, 30, 40);
@names = ("John Paul", "Lisa", "Kumar");

print "\$ages[0] = $ages[0]\n";
print "\$ages[1] = $ages[1]\n";
print "\$ages[2] = $ages[2]\n";
print "\$names[0] = $names[0]\n";
print "\$names[1] = $names[1]\n";
print "\$names[2] = $names[2]\n";
```

Here we used escape sign (\) before the $ sign just to print it. Other Perl will understand it as a variable and will print its value. When executed, this will produce the following result −

```
$ages[0] = 25
$ages[1] = 30
$ages[2] = 40
$names[0] = John Paul
$names[1] = Lisa
$names[2] = Kumar
```

Hash Variables

A hash is a set of **key/value** pairs. Hash variables are preceded by a percent (%) sign. To refer to a single element of a hash, you will use the hash variable name followed by the "key" associated with the value in curly brackets.

Here is a simple example of using hash variables −

Live Demo

```
#!/usr/bin/perl

%data = ('John Paul', 45, 'Lisa', 30, 'Kumar', 40);

print "\$data{'John Paul'} = $data{'John Paul'}\n";
print "\$data{'Lisa'} = $data{'Lisa'}\n";
print "\$data{'Kumar'} = $data{'Kumar'}\n";
```

This will produce the following result −

```
$data{'John Paul'} = 45
$data{'Lisa'} = 30
$data{'Kumar'} = 40
```

Variable Context

Perl treats same variable differently based on Context, i.e., situation where a variable is being used. Let's check the following example −

Live Demo

```
#!/usr/bin/perl

@names = ('John Paul', 'Lisa', 'Kumar');

@copy = @names;
$size = @names;

print "Given names are : @copy\n";
print "Number of names are : $size\n";
```

This will produce the following result −

```
Given names are : John Paul Lisa Kumar
Number of names are : 3
```

Here @names is an array, which has been used in two different contexts. First we copied it into anyother array, i.e., list, so it returned all the elements assuming that context is list context. Next we used the same array and tried to store this array in a scalar, so in this case it returned just the number of elements in this array assuming that context is scalar context.

Perl - CGI Programming

What is CGI ?

- A Common Gateway Interface, or CGI, is a set of standards that defines how information is exchanged between the web server and a custom script.
- The CGI specs are currently maintained by the NCSA and NCSA defines CGI is as follows −

- *The Common Gateway Interface, or CGI, is a standard for external gateway programs to interface with information servers such as HTTP servers.*
- The current version is CGI/1.1 and CGI/1.2 is under progress.

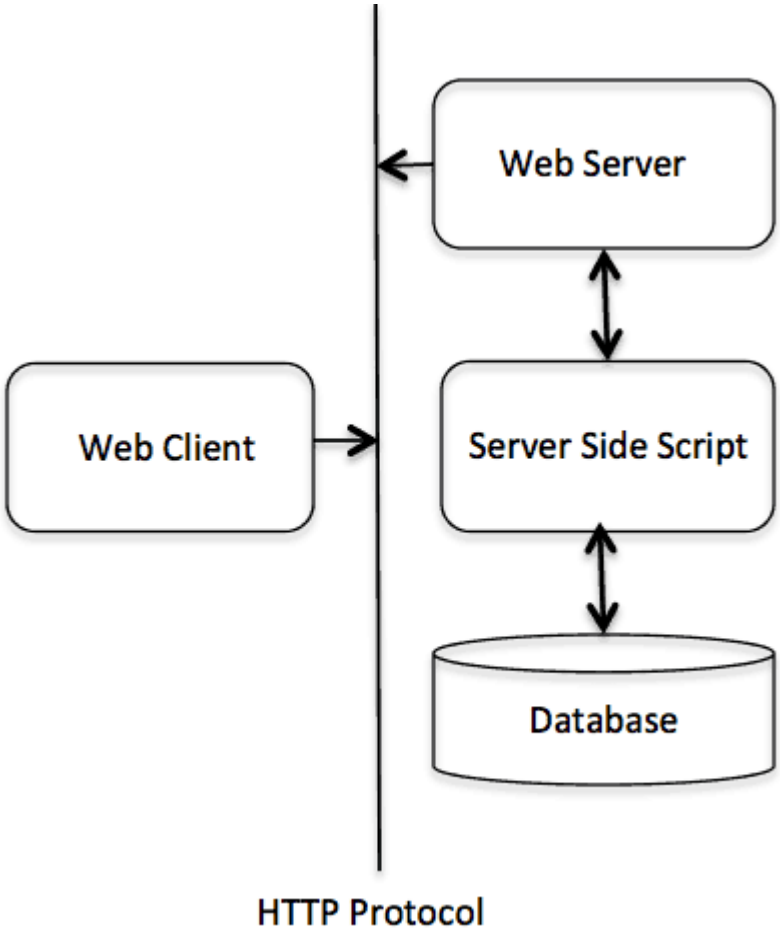
Web Browsing

To understand the concept of CGI, lets see what happens when we click a hyper link available on a web page to browse a particular web page or URL.

- Your browser contacts web server using HTTP protocol and demands for the URL, i.e., web page filename.
- Web Server will check the URL and will look for the filename requested. If web server finds that file then it sends the file back to the browser without any further execution otherwise sends an error message indicating that you have requested a wrong file.
- Web browser takes response from web server and displays either the received file content or an error message in case file is not found.

However, it is possible to set up HTTP server in such a way so that whenever a file in a certain directory is requested that file is not sent back; instead it is executed as a program, and whatever that program outputs as a result, that is sent back for your browser to display. This can be done by using a special functionality available in the web server and it is called **Common Gateway Interface** or CGI and such programs which are executed by the server to produce final result, are called CGI scripts. These CGI programs can be a PERL Script, Shell Script, C or C++ program, etc.

CGI Architecture Diagram



Web Server Support and Configuration

Before you proceed with CGI Programming, make sure that your Web Server supports CGI functionality and it is configured to handle CGI programs. All the CGI programs to be executed by the web server are kept in a pre-configured directory. This directory is called CGI directory and by convention it is named as /cgi-bin. By convention Perl CGI files will have extention as **.cgi**.

First CGI Program

Here is a simple link which is linked to a CGI script called hello.cgi. This file has been kept in **/cgi-bin/** directory and it has the following content. Before running your CGI program, make sure you have change mode of file using **chmod 755 hello.cgi** UNIX command.

```
#!/usr/bin/perl

print "Content-type:text/html\r\n\r\n";
print '<html>';
print '<head>';
print '<title>Hello Word - First CGI Program</title>';
print '</head>';
print '<body>';
```

```
print '<h2>Hello Word! This is my first CGI program</h2>';
print '</body>';
print '</html>';

1;
```

Now if you click **hello.cgi** link then request goes to web server who search for hello.cgi in /cgi-bin directory, execute it and whatever result got generated, web server sends that result back to the web browser, which is as follows −

Hello Word! This is my first CGI program

This hello.cgi script is a simple Perl script which is writing its output on STDOUT file, i.e., screen. There is one important and extra feature available which is first line to be printed **Content-type:text/html\r\n\r\n**. This line is sent back to the browser and specifies the content type to be displayed on the browser screen. Now you must have undertood basic concept of CGI and you can write many complicated CGI programs using Perl. This script can interact with any other exertnal system also to exchange information such as a database, web services, or any other complex interfaces.

Understanding HTTP Header

The very first line **Content-type:text/html\r\n\r\n** is a part of HTTP header, which is sent to the browser so that browser can understand the incoming content from server side. All the HTTP header will be in the following form −

HTTP Field Name: Field Content

For Example −

```
Content-type:text/html\r\n\r\n
```

There are few other important HTTP headers, which you will use frequently in your CGI Programming.

Sr.No.	Header & Description
1	**Content-type: String** A MIME string defining the format of the content being returned. Example is Content-type:text/html
2	**Expires: Date String** The date when the information becomes invalid. This should be used by the browser to decide when a page needs to be refreshed. A valid date string should be in the format 01 Jan 1998 12:00:00 GMT.
3	

	Location: URL String The URL that should be returned instead of the URL requested. You can use this filed to redirect a request to any other location.
4	**Last-modified: String** The date of last modification of the file.
5	**Content-length: String** The length, in bytes, of the data being returned. The browser uses this value to report the estimated download time for a file.
6	**Set-Cookie: String** Set the cookie passed through the *string*

CGI Environment Variables

All the CGI program will have access to the following environment variables. These variables play an important role while writing any CGI program.

Sr.No.	Variables Names & Description
1	**CONTENT_TYPE** The data type of the content. Used when the client is sending attached content to the server. For example file upload, etc.
2	**CONTENT_LENGTH** The length of the query information. It's available only for POST requests
3	**HTTP_COOKIE** Returns the set cookies in the form of key & value pair.
4	**HTTP_USER_AGENT** The User-Agent request-header field contains information about the user agent originating the request. Its name of the web browser.
5	**PATH_INFO** The path for the CGI script.

6	**QUERY_STRING** The URL-encoded information that is sent with GET method request.
7	**REMOTE_ADDR** The IP address of the remote host making the request. This can be useful for logging or for authentication purpose.
8	**REMOTE_HOST** The fully qualified name of the host making the request. If this information is not available then REMOTE_ADDR can be used to get IR address.
9	**REQUEST_METHOD** The method used to make the request. The most common methods are GET and POST.
10	**SCRIPT_FILENAME** The full path to the CGI script.
11	**SCRIPT_NAME** The name of the CGI script.
12	**SERVER_NAME** The server's hostname or IP Address.
13	**SERVER_SOFTWARE** The name and version of the software the server is running.

Here is a small CGI program to list down all the CGI variables supported by your Web server. Click this link to see the result Get Environment

```
#!/usr/bin/perl

print "Content-type: text/html\n\n";
print "<font size=+1>Environment</font>\n";
foreach (sort keys %ENV) {
   print "<b>$_</b>: $ENV{$_}<br>\n";
}
```

```
1;
```

Raise a "File Download" Dialog Box?

Sometime it is desired that you want to give option where a user will click a link and it will pop up a "File Download" dialogue box to the user instead of displaying actual content. This is very easy and will be achived through HTTP header.

This HTTP header will be different from the header mentioned in previous section. For example, if you want to make a **FileName** file downloadable from a given link then it's syntax will be as follows −

```
#!/usr/bin/perl

# HTTP Header
print "Content-Type:application/octet-stream; name = \"FileName\"\r\n";
print "Content-Disposition: attachment; filename = \"FileName\"\r\n\n";

# Actual File Content will go hear.
open( FILE, "<FileName" );
while(read(FILE, $buffer, 100) ) {
   print("$buffer");
}
```

GET and POST Methods

You must have come across many situations when you need to pass some information from your browser to the web server and ultimately to your CGI Program handling your requests. Most frequently browser uses two methods to pass this information to the web server. These methods are **GET** Method and **POST** Method. Let's check them one by one.

Passing Information using GET Method

The GET method sends the encoded user information appended to the page URL itself. The page and the encoded information are separated by the ? character as follows −

http://www.test.com/cgi-bin/hello.cgi?key1=value1&key2=value2

The GET method is the defualt method to pass information from a browser to the web server and it produces a long string that appears in your browser's Location:box. You should never use GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be passed in a request string.

This information is passed using **QUERY_STRING** header and will be accessible in your CGI Program through QUERY_STRING environment variable which you can parse and use in your CGI program.

You can pass information by simply concatenating key and value pairs alongwith any URL or you can use HTML <FORM> tags to pass information using GET method.

Simple URL Example: Get Method

Here is a simple URL which will pass two values to hello_get.cgi program using GET method.

http://www.tutorialspoint.com/cgi-bin/hello_get.cgi?first_name=ZARA&last_name=ALI

Below is **hello_get.cgi** script to handle input given by web browser.

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;
if ($ENV{'REQUEST_METHOD'} eq "GET") {
   $buffer = $ENV{'QUERY_STRING'};
}
# Split information into name/value pairs
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}
$first_name = $FORM{first_name};
$last_name  = $FORM{last_name};

print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Hello - Second CGI Program</title>";
print "</head>";
print "<body>";
print "<h2>Hello $first_name $last_name - Second CGI Program</h2>";
print "</body>";
print "</html>";

1;
```

Simple FORM Example: GET Method

Here is a simple example, which passes two values using HTML FORM and submit button. We are going to use the same CGI script hello_get.cgi to handle this input.

```
<FORM action = "/cgi-bin/hello_get.cgi" method = "GET">
First Name: <input type = "text" name = "first_name">  <br>

Last Name: <input type = "text" name = "last_name">
```

```
<input type = "submit" value = "Submit">
</FORM>
```

Here is the actual output of the above form coding. Now you can enter First and Last Name and then click submit button to see the result.

First Name:

Last Name: Submit

Passing Information using POST Method

A more reliable method of passing information to a CGI program is the **POST** method. This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a **?** in the URL, it sends it as a separate message as a part of HTTP header. Web server provides this message to the CGI script in the form of the standard input.

Below is the modified **hello_post.cgi** script to handle input given by the web browser. This script will handle GET as well as POST method.

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;
if ($ENV{'REQUEST_METHOD'} eq "POST") {
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
} else {
   $buffer = $ENV{'QUERY_STRING'};
}
# Split information into name/value pairs
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}
$first_name = $FORM{first_name};
$last_name  = $FORM{last_name};

print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Hello - Second CGI Program</title>";
print "</head>";
print "<body>";
print "<h2>Hello $first_name $last_name - Second CGI Program</h2>";
print "</body>";
```

```
print "</html>";

1;
```

Let us take again same examle as above, which passes two values using HTML FORM and submit button. We are going to use CGI script hello_post.cgi to handle this input.

```
<FORM action = "/cgi-bin/hello_post.cgi" method = "POST">
First Name: <input type = "text" name = "first_name">  <br>

Last Name: <input type = "text" name = "last_name">

<input type = "submit" value = "Submit">
</FORM>
```

Here is the actual output of the above form coding, You enter First and Last Name and then click submit button to see the result.

First Name:

Last Name: Submit

Passing Checkbox Data to CGI Program

Checkboxes are used when more than one option is required to be selected. Here is an example HTML code for a form with two checkboxes.

```
<form action = "/cgi-bin/checkbox.cgi" method = "POST" target = "_blank">
<input type = "checkbox" name = "maths" value = "on"> Maths
<input type = "checkbox" name = "physics" value = "on"> Physics
<input type = "submit" value = "Select Subject">
</form>
```

The result of this code is the following form −

Maths Physics Select Subject

Below is **checkbox.cgi** script to handle input given by web browser for radio button.

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;
if ($ENV{'REQUEST_METHOD'} eq "POST") {
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
} else {
   $buffer = $ENV{'QUERY_STRING'};
}
```

```
# Split information into name/value pairs
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}
if( $FORM{maths} ) {
   $maths_flag ="ON";
} else {
   $maths_flag ="OFF";
}
if( $FORM{physics} ) {
   $physics_flag ="ON";
} else {
   $physics_flag ="OFF";
}

print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Checkbox - Third CGI Program</title>";
print "</head>";
print "<body>";
print "<h2> CheckBox Maths is : $maths_flag</h2>";
print "<h2> CheckBox Physics is : $physics_flag</h2>";
print "</body>";
print "</html>";

1;
```

Passing Radio Button Data to CGI Program

Radio Buttons are used when only one option is required to be selected. Here is an example HTML code for a form with two radio button −

```
<form action = "/cgi-bin/radiobutton.cgi" method = "POST" target = "_blank">
<input type = "radio" name = "subject" value = "maths"> Maths
<input type = "radio" name = "subject" value = "physics"> Physics
<input type = "submit" value = "Select Subject">
</form>
```

The result of this code is the following form −

Below is **radiobutton.cgi** script to handle input given by the web browser for radio button.

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

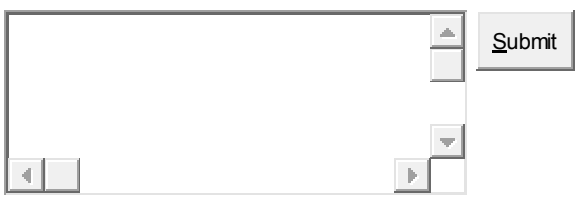
```
local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;
if ($ENV{'REQUEST_METHOD'} eq "POST") {
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
} else {
   $buffer = $ENV{'QUERY_STRING'};
}
# Split information into name/value pairs
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}
$subject = $FORM{subject};

print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Radio - Fourth CGI Program</title>";
print "</head>";
print "<body>";
print "<h2> Selected Subject is $subject</h2>";
print "</body>";
print "</html>";

1;
```

Passing Text Area Data to CGI Program

A textarea element is used when multiline text has to be passed to the CGI Program. Here is an example HTML code for a form with a TEXTAREA box −

```
<form action = "/cgi-bin/textarea.cgi" method = "POST" target = "_blank">
<textarea name = "textcontent" cols = 40 rows = 4>
Type your text here...
</textarea>
<input type = "submit" value = "Submit">
</form>
```

The result of this code is the following form −



Below is the **textarea.cgi** script to handle input given by the web browser.

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;
if ($ENV{'REQUEST_METHOD'} eq "POST") {
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
} else {
   $buffer = $ENV{'QUERY_STRING'};
}
# Split information into name/value pairs
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}
$text_content = $FORM{textcontent};

print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Text Area - Fifth CGI Program</title>";
print "</head>";
print "<body>";
print "<h2> Entered Text Content is $text_content</h2>";
print "</body>";
print "</html>";

1;
```