



# Newsletter #4

# Table of Contents

Editorial  
by MsMittens .....p. 3

A Beginner's Guide to PHP  
by chsh .....p. 4

The Gimp  
by Andrew J. Bennieston .....p. 13

So you wanna be a H4x0r?  
by Terr .....p. 19

Installing Apache on Windows  
by Will W. ....p. 22

C++ Programming (Chapter 1)  
by Jethro .....p. 24

C++ Programming (Chapter 2)  
by Jethro .....p. 29

Another Beginner's Guide to PHP  
by Rewandythal .....p.34

# MsMittens' Editorial

How ironic it is that in the last issue of the Newsletter I commented on how quiet things have gotten. And lo and behold we see since then a new Windows worm along the same flavour as Code Red. In fact, this one managed to infect 7,000 machines in a 24 hour period. And much like it's ancestor, it was created to target a single site: the Pakistan Government's Website.

In addition, Klez is going strong and variants of it are popping up everywhere. In fact, I myself received at least 6 emails in a 24 hour period that contained Klez and/or variants thereof. It is one of the few times I am glad I use a Macintosh to collect email. ;D

But overall, it would appear that the famed attacks found in the early-mid 90s have gone away. With them have gone the Fine Art of Hacking, in the true sense of the concept. We see today numerous scriptkiddies who are unwilling to learn beyond the simple DoS type attacks. With many post-secondary environments teaching potential network administrators how to look after networks but never worry about scripting or programming ("because hey.. yer just an admin. You never need that stuff"), I wonder what our future of IT will look like 10 years down the road.

Theoretically, we will be using IPv6 and have Internet-based Operating Systems. We will have machines that can run at a 1000% faster than present day and few applications will match it. And server operating systems will continue to be made for the simplest of minds. A true distinction between networker and programmer will be made. This however is not necessarily a good thing.

A lot of talent and flexibility is lost when we don't expand our knowledge beyond the quick and easy. We loose our curiosity because everything we want is given to us on a silver platter and/or created by someone. In this issue, a lot of creativity is evident as we see not one but two beginner's guides to PHP. Both provide good information from two points of view and are unique in their presentation. They also show the flexibility and power that can be found with this language. You will also find a look at C++ and Gimp as well as Apache on Windows.

Last but not least is Terr's advice on being a "h4x0r". Enjoy and remember, you can be a scriptkiddies or you can be a god (or goddess). It is all a matter of creativity and curiosity. Enjoy.

P.S. I was at SecurityFocus recently and they listed their top Ten for attacks and vulnerabilitites. Remember, the more you know, the better you can secure your systems.

SecurityFocus's Top 10 Attacks for 2002 First Quarter:[http://www.securityfocus.com/corporate/research/top10attacks\\_q1\\_2002.shtml](http://www.securityfocus.com/corporate/research/top10attacks_q1_2002.shtml)

SecurityFocus's Top 10 Vulnerabilities for 2002 First Quarter:[http://www.securityfocus.com/corporate/research/top10vulns\\_q1\\_2002.shtml](http://www.securityfocus.com/corporate/research/top10vulns_q1_2002.shtml)

# A Beginner's guide to PHP.

## Part 1: The Very Basics by chsh

I'd like to preface this article with a few words on where I'm coming from in the programming experience department. I'm employed currently full time as an

ASP and PHP web developer. Most of my work involves connecting to some kind of database. I learned PHP more as a hobby at home, and then began to see its

potential use for work, and applied it there. I knew Visual Basic (and by extension, VBScript) and JavaScript, and had begun to learn C, which gave me a tremendous advantage in the basic syntax. If you do not know JavaScript or C, then don't worry, as this is the beginner's tutorial, I'm aiming it at people with little programming experience.

Now, as they say, on with the show.

### Section A: Basic Syntax

---

PHP is a C-style language. Its basic syntax is almost identical to C, so if you know C, you can probably skip to section H (most of the way through this

tutorial). For those of you who don't know C, or any similar language (Java, JavaScript, C++, etc), read on.

The first place to start with PHP is with the end of line character. PHP reads a line of code until it encounters a semicolon (;), at which point it processes

the statement before that semicolon, and then proceeds on to the next statement.

An example:

```
statement number 1;
```

```
statement number 2;
```

Like most languages, the single equals sign (=) is used to set a variable. Variables in PHP are all prefixed with the dollar sign (\$).

An example:

```
$somevar = $someothervar;
```

Comments in your code are done using double forward slash (//) to comment out a single line, and a forward slash and asterisk combination (/\*) to comment out a

block of text on multiple lines. To close out a multi-line comment, you use a asterisk and forward slash combination (\*/).

An example:

```
// This is a single line comment
```

```
/* This is a multi-line comment.
```

```
This is a multi-line comment. */
```

Incrementing/decrementing any variable (integer or character) is done using the double plus (++) or double minus (--) operators right after the variable is used.

An example:

```
$i = 0; // i is equal to zero.
```

```
$i++; // i is set to one, after this piece of code is executed.
```

```
$i--; // i is set to zero again, after this piece of code is executed.
```

```
++$i; // i is set to one again, this time before this piece of code is executed.
```

```
--$i; // i is set to zero again, this time before this peice of code is executed.
```

Section B: If, Elseif, and Else

---

Conditional operations are done use the If/Elseif/Else statement. It would normally look like the following:

An example of a standard IF statement:

```
if ($somevar == 1) {  
    $somevar = $someothervar;  
}
```

An english interpretation of how this would read:

If the value of the somevar variable is equal to 1, then set the somevar variable to the value of the someothervar variable.

An example of a standard IF/ELSE statement:

```
if ($somevar == 1) {  
    $somevar = $someothervar;  
} else {  
    $somevar = $someothervar + 1;  
}
```

An english interpretation of how this would read:

If the value of somevar is equal to 1, then set somevar to the value of someothervar, otherwise set somevar to someothervar plus one.

An example of a standard IF/ELSEIF/ELSE statement:

```
if ($somevar == 1) {  
    // process some stuff  
    $somevar = $someothervar;  
} elseif ($somevar == 2) {  
    // process some other stuff  
    $somevar = $someothervar - 1;
```

```
} else {  
    // process any other options  
    $somevar = $someothervar + 1;  
}
```

An english interpretation of how this would read:

If the value of somevar is equal to 1, then set somevar to the value of someothervar, otherwise if somevar is equal to 2, then set somevar equal to

someothervar minus one, otherwise set somevar to someothervar plus one.

Conditions are evaluated between round brackets (), and the action to take if a specified condition is met is identified by the curly braces {}. If you have a

single line in the action section of a conditional statement, you can simply ignore the curly braces, however I find it is much easier to debug code if you

put all the braces in.

An example of a standard IF statement with one line of action:

```
if ($somevar == 1)  
    $somevar = $someothervar;
```

The above will execute the same way the standard IF statement shown above it will execute.

### Section C: Comparison Operators

---

PHP uses various comparison operators. A brief rundown of them is:

== Equal: Returns true if both variables are equal.

=== Identical: Returns true if both variables are equal, and of the same data type.

!= Not Equal: Returns true if both variables are not equal.

<> Same as Not Equal.

!== Not Identical: Returns true if both variables either either not equal, or not of the same data type. (This is available in PHP versions 4+ only)

< Less than: Returns true if the first operator is less than the second operator.

> Greater than: Returns true if the first operator is greater than the second operator.

<= Less than or Equal to: Returns true if the first operator is less than or equal to the second operator.

>= Greater than or Equal to: Returns true if the first operator is greater than or equal to the second operator.

#### Section D: While Loops

---

While loops are a fairly common method of looping until you reach a certain point in processing.

The syntax is as follows:

```
while (condition) {  
    // do stuff  
}
```

An example:

```
while ($foundit == false) {  
    if ($temp == 1) {  
        $foundit == true;  
    }  
    $temp = $newval;  
}
```

An english interpretation of how this would read:

While foundit is equal to false, check if temp is equal to 1. If it is equal to 1, set foundit to true. Set temp to newval.

---

Section F: The Switch Statement

-----

The switch statement works much the same as the If statement, but is much more geared towards large lists of comparisons. A switch statement can save you a lot of code writing if properly employed.

The syntax is as follows:

```
switch (var) {  
    case 1:  
        $varIsOne = true;  
    case 2:  
        $varIsTwo = true;  
    default:  
        $varUnknown = true;  
}
```

An english interpretation of how this would read:

Comparing var. If it is one, set varIsOne to true. If it is two, set varIsTwo to true. If it does not match any of the criteria, set varUnknown to true.

Section G: For Loops

---

The For loop is possibly the single most useful loop you will find. The for loop is designed to increment a counter (or run a piece of code) on each

iteration of the loop, making it ideal for working with arrays, databases, etc.. The For loop takes three arguments, separated by semicolon: the

first, the value to start with. The second, the value to end at, and the third, the piece of code to execute in order to increment the variable.

An example:

```
for ($i=0; $i<$count; $i++) {  
    // Do stuff here.  
    $temp = $i;  
}
```

An english interpretation of how this would read:

I is post-incrementing from 0, to less than count. While it is incrementing, set temp equal to i.

## Section H: Arrays

---

Arrays are another useful feature of many programming languages. They allow you to store similar data in a variable, but reference it by a key. In the case of

PHP, the key can be an integer, or a string of text. An array's key is passed in square brackets.

An example:

```
Animal[0] = "Dog";  
Animal[1] = "Cat";  
Animal[2] = "Fish";
```

An english interpretation of how this would read:

Set element zero in the Animal array to "Dog". Set element one in the Animal array to "Cat". Set element two in the Animal array to "Fish".

An example using a string key:

```
Leash["Dog"] = true;  
Leash["Cat"] = true;  
Leash["Fish"] = false;
```

An english interpretation of how this would read:

Set element "Dog" in the Leash array to true. Set element "Cat" in the Leash array to true. Set element "Fish" in the Leash array to false.

## Section I: PHP Built-In Variables and Arrays

---

PHP has a number of built-in variables. As this is a web scripting language, it contains variables that contain POST and GET data, as well as many other

variables, including the name of the current script, the name of the webserver being connected to, the version of PHP, etc., etc.. I will provide a brief list

here, but <http://www.php.net/manual/en/language.variables.predefined.php> is a much more complete and up to date list. I am intentionally using the older

versions of these array names, simply because while the PHP developers will be deprecating them, there are still a lot of PHP installations out there that are

less than version 4.1.0, which was when \$\_SERVER[] et. al. were introduced.

`$HTTP_SERVER_VARS[]` An array containing webserver paths, script names, etc.

`$HTTP_POST_VARS[]` An array containing all the data POSTed to the script.

`$HTTP_GET_VARS[]` An array containing all the query string data.

`$HTTP_COOKIE_VARS[]` An array containing all the data passed via cookies.

`$HTTP_SESSION_VARS[]` An array containing all session data on scripts where session information is being tracked.

## Section J: PHP Basic Built-In Functions

---

PHP, like most programming languages, comes with loads of functions. I'm going to go over only the very basic functions that allow you to write out text, as

well as some other simple functions you might have a use for.

`echo()` This function allows you to write text out. Example: `echo "Hi World."`

`print()` This function also allows you to write text out. It is slightly different in implementation than `echo`, but for most uses, `print` and `echo` are synonymous.

`strToUpper()` This function allows you to convert a variable to completely uppercase.

`strToLower()` This function allows you to convert a variable to completely lowercase.

ucfirst() This function converts the first character in the passed string to upper case. This is useful for beginning a paragraph.

substr() This function is a very advanced string slicing and dicing function. It allows you to grab part of the string passed to it based on the number of

characters you want, and the orientation you want them from. For example, if you want to retrieve the four leftmost characters from a string, substr can do that

for you.

These are just some simple common functions, and I have no doubt left something out of this. If you feel this needs major correction, please email me at

[chsh1ca@yahoo.ca](mailto:chsh1ca@yahoo.ca). I also encourage you to read the function library (for in depth descriptions of these functions and their uses) in the online PHP manual at

<http://www.php.net/manual/>

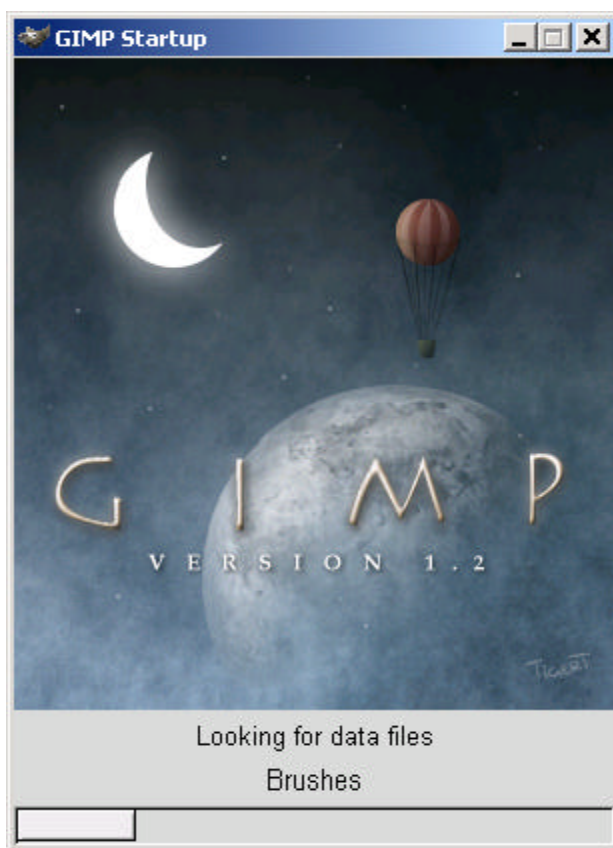
I'll be holding an IRC class on PHP on **AntiOnline's IRC** server on **Wednesday, July 10th, at 7pm EST**. If you wish to attend, please show up no later than

7:30pm. If you have any questions or comments, please feel free to email me.

Chsh

# The GIMP

## Features, Benefits, Drawbacks, And Comparisons With Other Image Editors (Win32 Edition)



**Andrew J. Bennieston**

# Table Of Contents

<a href="#">Table Of Contents</a>	.....
<a href="#">What Is The GIMP?</a>	.....
<a href="#">Where Can I Download It?</a>	.....
<a href="#">But Where's All The Functionality?</a>	.....
<a href="#">Main Features</a>	.....
<a href="#">Benefits Of the Win32 Version</a>	.....
<a href="#">Drawbacks Of Running The GIMP on Win32</a>	.....
<a href="#">Comparison: The GIMP Vs. MS PhotoDraw 2000</a>	.....
<a href="#">Comparison: The GIMP Vs. Paint Shop Pro</a>	.....
<a href="#">Comparison: The GIMP Vs. Adobe Photoshop</a>	.....
<a href="#">Documentation</a>	.....
<a href="#">About This Document</a>	.....

## What Is The GIMP?

The GIMP is a free image editor. It was originally designed to run on Unix/Linux systems (referred to from here on in as \*nix systems). These systems have many advantages over Windows, and we will not go into detail about how bad Windows really is in this article, since those interested in image editing may not actually want a computer that works well enough and fast enough to be of any use whatsoever.

The GIMP stands for The Gnu Image Manipulation Program. As the official website for the project states,

*“The GIMP is the GNU Image Manipulation Program. It is a freely distributed piece of software suitable for such tasks as photo retouching, image composition and image authoring. This site contains information about downloading, installing, using, and enhancing GIMP. This site also serves as a distribution point for the latest releases, patches, plugins, and scripts. We also try to provide as much information about the GIMP community and related projects as possible.”*

The GIMP Website can be found at <http://www.gimp.org>

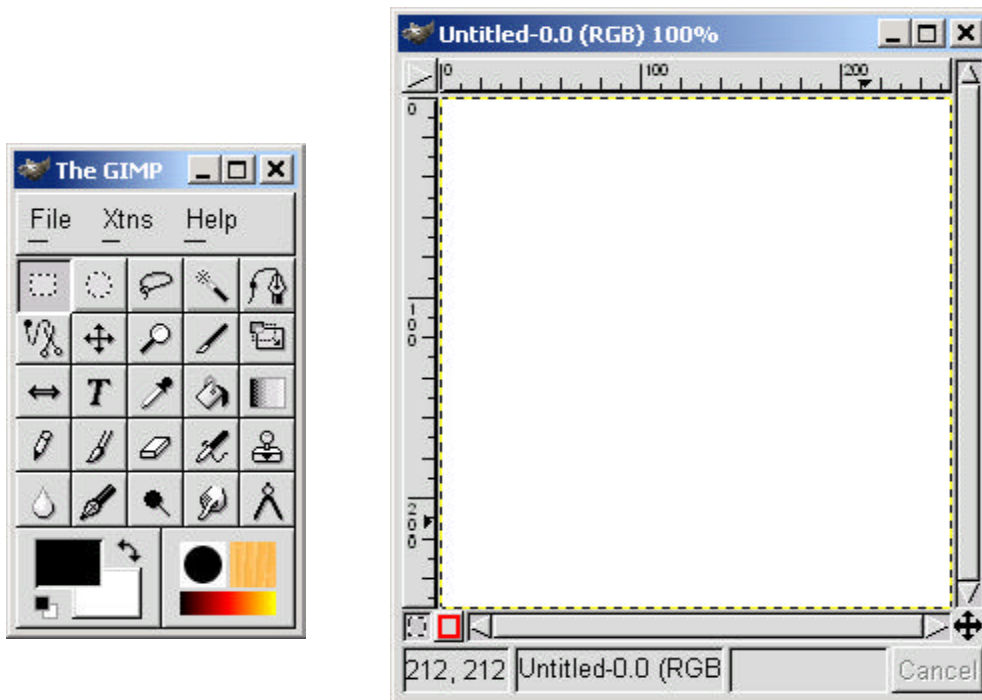
## Where Can I Download It?

The official website is usually the best place to download anything, you can download The GIMP from [www.gimp.org](http://www.gimp.org), although the link below takes you directly to the download:

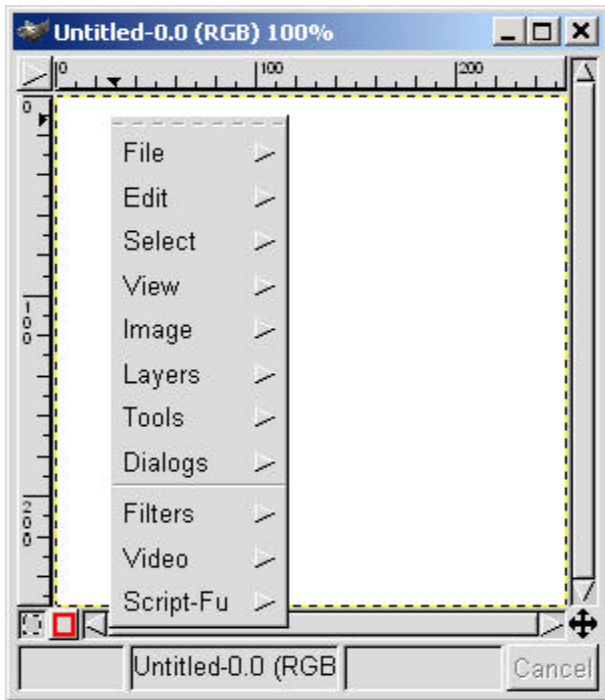
<http://www.gimp.org/~tml/gimp/win32/downloads.html>

## But Where's All The Functionality?

Once you have downloaded and installed The GIMP, the first thing you may notice is that once you have created a new image, there are no tools to use beyond the traditional paint tools. Or are there?



The tools and filters, save functionality, editing capabilities etc. are available on a right-click menu within the main image area:



This is where you can now explore The GIMP's true power and ability.

## Main Features

- Layers
- Compatible with JPG, PNG, BMP and numerous other file formats
- Assorted filters including rendering, light effects, distortions, enhancements etc.
- Zoom capability
- Editing capabilities including copying of images and parts thereof
- Save and Open capability for several file types
- Completely free software
- Video capabilities
- Customisable through Script-Fu plugins, additional brushes, gradients etc.

## Benefits Of the Win32 Version

The Win32 version of The GIMP has no advantages whatsoever over the \*nix version. It does, however, have one major advantage over other Win32 image editing software... it is completely, totally, 102% free.

## Drawbacks Of Running The GIMP on Win32

The Win32 version of The GIMP is slower than the \*nix version and is more prone to crashing. It has no other drawbacks, it is by far the best image editor for the Win32 platform.

## Comparison: The GIMP Vs. MS PhotoDraw 2000

The first point of comparison is price. The GIMP wins hands down on this, considering that it costs nothing, compared with the extortionate prices Microsoft charge for their software.

As far as functionality is concerned, The GIMP can do everything PhotoDraw can do, and more. PhotoDraw does make certain tasks easier for the user, for example adding and positioning a shadow is automated in PhotoDraw, although it is possible with great ease to get equal if not better effects from The GIMP.

PhotoDraw has one advantage over The GIMP (if it can be called an advantage). PhotoDraw can save GIF files, whereas The GIMP cannot, for licensing reasons (<http://burnallgifs.org>). Decide for yourself whether you really want to use GIF, when PNG has all the functionality and none of the legal ties.

## Comparison: The GIMP Vs. Paint Shop Pro

PSP is a long-respected Win32 image editor, although compared to The GIMP it is still infantile in its feature base.

## Comparison: The GIMP Vs. Adobe Photoshop

Photoshop is the only real contender with The GIMP. It has pretty much all the same functionality as The GIMP, but it does have a price tag... and not a cheap one either.

# Documentation

<http://www.gimp.org/docs.html>

Documentation includes:

- Core tools
- GIMP Manual
- Tutorial
- Installation Help

The GIMP Website has many resources and informative links, and anyone who uses The GIMP will most likely find the majority of the content very useful.

## About This Document

Author: Andrew J. Bennieston

Date: Thursday 30 May 2002

Time: 15:57 GMT

Local Time: 16:57 BST

Distribution:

AntiOnline, <http://www.antionline.com>

Firestorm UK, <http://www.firestormuk.cjb.net>

# So you want to be a H4x0r? by Terr

First, a few disclaimers. I am writing this only from my personal perspective. There are no statistics, no interviews, etc. Secondly, I feel I am in the obscure not-quite-quilted not-quite-unqualified group, in that A) I don't program for a living, and B) Spend a lot of time online. If I end up sounding like a dictionary, I apologize, although you might need one to read this. If you want to look up some word and can't find it in a dictionary, you can try to find it here: <http://www.antionline.com/jargon/> .

So you say you want to hack?

You might have heard your friends boasting about how they got into another friend's computer and read their mail, or how they are going to try and change their grades at school through the computer, and thought to yourself: "I want to do that."

Well, this text will not teach you how to "hack". If you want to know where to download something, it probably won't be mentioned here. This text is how to NOT appear childish and immature in the online security community. This text consists of a few rules of etiquette (manners), some misconception-corrections (things that aren't what you think), and words of warning.

Do not use "l33t \$p33k" or excessive slang.

Many times this simply makes it harder for people to understand you, and no matter how cool it might sound in junior high, it's not. You will be quickly branded a "script kiddie", which is a term for a childish character who tries to make themselves look skilled by using programs and exploits (more on those later) already made by other people. Trust me, it's like being a called a wannabe.

When you have a problem or a question, be detailed!

Many times people will refuse to help because they don't want to be tricked into working as technical support for a clueless person. If you have a problem, you must state the problem clearly, and give as much information as you can about the problem that is relevant, otherwise people will choose to ignore you and hope you go away, because they don't want to spend half their time teaching you things in order to get the job done. It's like asking a mechanic: "There's a banging sound in my car, I'm not sure what kind of car I have, when it happens, or where it sounds like it is coming from.". The mechanic will probably be very frustrated when you can't tell them anything else.

When asking for help, don't expect a super-detailed answer.

Many 'how do you' requests are complex! There is no simple 10-step plan to doing it. It is a bit like asking someone how they get to a certain town in another country that they've never been to. They have some idea of what to expect, but there are so many possible routes and possible problems that they can only give advice, not always instructions. What if the plane is late? What if you only have a certain amount of money? What if there is a rock-slide in this mountainous area, how do you go around? In short, there are no simple answers, don't expect them.

Do not expect people to help you “hack”.

Remember, many requests, such as “I want to hack Hotmail.”, would actually get the person helping you in trouble, if you were to be caught! It’s like asking people to help you steal from a store, most of them might give you some information, just to be nice, but very few of them will really get involved.

Speak (and write) clearly!

I can’t stress this enough. A vast majority of the technically-skilled people on the internet happen to know how to read and write fairly well; the communication needed in their jobs requires it. People tend to get a bit annoyed when they receive a message which takes them a while to understand because it is written so badly. Be short, simple, to the point. However, remember, when you have a problem, it is better to give more information than none.

Hackers and Crackers.

This is just a warning section. Some people, mostly the media, talk about people breaking into computer systems and call them “Hackers”. Originally, the words “Hacker” and “Hacking” did NOT mean something bad, but it’s become like that due to the way the media tends to use shocking headlines. (For the linguistically-interested, there are other examples, a Villian is a ‘person of the villa’, or a peasant. Sinister, in latin, means ‘left handed’.)

Many people believe that a “Hacker” is someone with technical skill and a way of thinking outside of the box in order to get results. The astronauts in Apollo 13 who managed to fix up their air-filtration systems (and the engineers on the Earth who helped find out how) would be hackers, in that they did a clever jury-rigged (messy but working and very cool) job. Fixing the system would be a hack. Some take it even farther: Getting a car on top of a school roof without anyone noticing or finding out how they did it would be a hack.

Now, “Crackers” are the bad guys. A cracker (Criminal + Hacker = Cracker) would be the people who steal credit-card numbers and do the mean things. To be safe and to not offend someone on the web, it’s probably safer to use Hacker and Cracker as separate groups. To use a geeky reference, the Hackers are the Jedi, the Crackers are the dark Sith (such as Darth Vader).

Some people have written that, in truth, Hacker has always been a negative word. Well, whatever the case, just be aware that both sides to the issue exist.

“Hacktivism”, or defacing websites

Ever heard of the term ‘hacktivism’? Apparently someone got that idea that they could do the equivalent of spray-painting the side of a building or breaking it’s windows, and then say that it was okay to do because they had a message to express. A great majority of website defacements (breaking in and changing the page) are infantile attempts to appear macho, not serious attempts to make an issue known. Moreover, even when there IS a real political reason behind it, it’s still not legal or ethical. It’s like trying to say: “Vote Bob!” in an election by breaking windows in an office building in order to spell out his name. In addition, many website defacements with political messages are stupid, because they deface the

website of a washing-machine manufacturer in order to tell about violence in Israel, or something. It doesn't make any sense, and it targets a totally unrelated bystander just because their web-server happens to be vulnerable to a script-kidde exploit.

Just remember, most people will not pat you on the back for defacing a website, instead they will probably see you as the internet-equivalent of a punk with a spray-paint can and big mouth.

I want to be undetectable online.

You may have heard of IP-spoofing or using proxies, and you may want to try to be 'invisible' online. To avoid going into overly-technical exceptions and explanations, the short answer is that you can't. If you want to do some evil deed and wreck somebody's website, just remember that somewhere, somehow, there is information that can lead people back to your name, address, telephone number, etc. The real question is how easy it is for them to get that information and how determined they are. If you choose to deface a website, you have to be prepared to face the music if it comes. Even using a proxy isn't fool-proof, because the proxy owner will probably have records which will allow them to figure out who was doing what through their server and when. "Anonymity" online largely involves having other people agree to help you stay anonymous, such as your ISP. If the police or FBI call your ISP, your ISP will probably decide that, in this case, they have to tell them the name of the person using a particular ISP account at a certain time, etc.

To make a long story simple and short, you cannot be 'invisible', and if the right people want to find you and they want it enough, you've got problems.

When trying to impress others.

If you're feeling like bragging about something, be forewarned. A great many people in security chat rooms and boards don't really care if it doesn't affect them. Some people will get annoyed at your attitude if they are aware that it was possible to do whatever you did with a pre-compiled program that someone else did most of the work for. Also, if you choose to go the 'black hat' route and become a cracker (which I don't recommend, see preceding section), remember that if you decide to brag, you drastically increase your chances of getting caught. So if you want to be bad in order to be (in-)famous, it's a pretty foolish idea.

# Installing Apache on Windows.

## by Will W.

Installing Apache on Windows, why? Because let's face it Windows is easy, and well Apache sure beats using IIS. This tutorial is meant for the person who would like to set up there own little web server. It's not meant for the IT Person running a fortune 500 company. But hey if you want go ahead.

Instalation:

First thing you need is to download the webserver. Now for windows users your gonna want to go download the .exe . The apache website is [www.apache.org](http://www.apache.org) Your gonna wanna head to the apache binaries sections for Win32 I believe it is at <http://www.apache.org/dist/httpd/binaries/win32/> There you will be able to download a version of apache.

Now before you download it you gonna want to make a folder. This folder is where your gonna server your root directory. Now if you don't want to do this it's ok. You can use the default path if you want. Put usually this helps in setting up other things like php, and MySQL. Most people do is they create a folder in the C:\ directory called WWW or somthin. You can name it whatever you want.

Ok so have downloaded the Apache Web Server. Your ready to go with the setup. No the version I have downloaded was `apache_2.0.36-win32-x86-no_ssl.msi` This was a newer version and supposedly supposed to be more secure. The first screen you get when your in the setup is The welcome screen we don't care much about that but owell so hit next. The next screen is the terms and service. And yes your going to agree to the terms duh. The next screen is some documentation. I never really read it but if you want go ahead and do it. Once your done hit next again. Know we see a screen that says enter a network domain. Erase what is ever in there and type localhost. Now the next box says Servername, erace what is ever in the box and put in localhost. The next is Administrators e-mail address. Go ahead and fill that in. But make sure to change it. Now there are 2 little radio buttons. Pick the one that best suites your needs. Now that we got that all filled out. Hit Next and you'll go to a screen that asks you which type of install you want to do. Then hit next.

If you wanted to server out of your one special folder. Change the file location of were your gonna install apache. Or just leave it at the default path. Click install and it should be on it's way. Once it's done installing hit the finish button.

The test:

First were gonna check to see if Apache installed correctly. This is how we do it. Open up Internet Explorer and type in " `http://localhost`" . If everything went smooth then you should be seeing a message that looks like this" Seeing this instead of the website you expected?" Yippee!!! Apache is working. See now wasnt' that really simple. Ok now were gonna do some fun stuff.

Alright now that we got or test done lets move on to changing some of this stuff that apache did on default. In Internet Explorer if you installed on the deafult path. Make your way to C:\Program Files\Apache Group\Apache2This is your Main Apache Directory were you can find everything. If you want take a short break and run around. There are some cool things there. Don't worry if you don't understand what's in these files just yet.

Break Time:

Go take a leak, get some pepsi and somthin to eat. If you got smokes light them up in your new found glory.

Alright so now you've got apache installed and your about to start dishing out your web pages that you took so much time on to build. Head to the folder called htdocs, this is your main folder. There should be a whole bunch of pages What i do is i select them all and move them to another folder. The htdocs folder is the best folder in the world. It's gonna be one of the places you spend most of your time dishing out content for the world. Ok so get rid of all that stuff that is in your htdocs folder. And move all your great content inside replacing it. Alright so now once we moved all are content inside the htdocs folder and we tested it to make sure it was there. <http://localhost> remember. Now let's get out of there. Go to Apache's main directory. Now just to be aware of what is going on and get a good example of how Apache Functions head off to a folder called "conf" This is the configuration files Apache Uses. If you ever wanted to install php and other server side scripting languages this is where you would do it. Now you get 2 copies Use 1 as a backup and never edit it at all. Go ahead and open the folder and open "httpd.conf" Read it very carefully cause in this tutorial were not gonna read about it. I just want you to know it's there. Anytime you edit the httpd.conf file you must re-start apache in order for it to work. Another good tip for you new people to apache is you may notice the log files. Yes there great and make sure to make backups of the logs they will come in handy. As security precautions. I also recommend getting a firewall set up. There are lots of great security features that apache has but this is a tutorial to installing apache.

Alright so now you've got your webpages up. But the only way people will be able to view your pages is my typing in your ip address. This is a bumper. Lets look at some free re-directories. [www.n2v.net](http://www.n2v.net), This is a cool one. You sign up put your ip adress of your new webserver in and whalla your done. Type in [www.n2v.net](http://www.n2v.net) and it goes to your server and brings up your super nice webpages. Now if you go to google and search for free domain names or re-directors you should come up with alot. Many People already know about the [www.dot.tk](http://www.dot.tk) one of the coolest things in the world. Free .tk very simple That's all you need. It works perfect for my webserver and I've got around 3,000 hits so it's working good. If you don't wanna do it you don't have to. But it just makes it simple.

Alright that comes to the conclusion of installing Apache Win32 for WINDOWS users. Very easy. One last thing Please Read more of the Apache Documentation either on there website or in your Apache2 directory. If you liked reading this tutorial on how to setup Apache check my website for others at [www.bonfire.tk](http://www.bonfire.tk) . Yes there will be follow ups. I'll be writing another apache tutorail soon so you can set up PHP. The most awesome scripting language ever built. And also another on how to secure Apache and yes ALL FOR WINDOWS!! .

# C++ Programming (Chapter One)

## by Jethro

**Topic: Programming**

<http://jethrojones.hyperlinx.cz>

-----

This tutorial is about basic programming with C++. Very basic. If you are a newbie however, you may find this tutorial quite useful... We cover basics like variables, functions, compilers, headers, operators...etc

C++ (pronounced "see-plus-plus") is an extended version of the C language which was developed in Bell Labs in 1978. In the early 80's, C++ was created as an object-oriented language. Without going into too much detail on the history of C++, I'll tell you this: C++ is the language which most software developers favour, because of its power, ease-of-use and its acceptance on most systems, such as Windows and \*NIX.

To make a C++ program you will need a compiler. I use Dev-C++ for Windows (<http://www.bloodshed.net>) but there are good ones such as Borland C++ Builder 5.5 (<http://www.borland.com>) and for UNIX, the famous GCC compiler is the best. This comes with a lot of UNIX systems like Red Hat Linux, I believe.

Before the code can be executed, it must be linked to other pieces of code (e.g. included libraries) used by the program. The compiled & linked program is called an executable file. Finally, the program is executed by the system. The compiler does this. Compilers take your typed source code and translate it into machine code (1s and 0s).

Here is an example program:

```
***** helloworld.cpp *****
```

```
/* This is a comment  
Hello World program  
Written by Jethro */
```

```
#include <iostream.h>
```

```
int main()  
{  
    cout << "Hello World" << endl; // First statement  
    return 0; // Second statement  
}
```

```
*****
```

Okay, time to go through this line-by-line...

```
*****
```

```
/* This is a comment
```

```
   Hello World program
```

Written by Jethro \*/ = This is a comment. There are two types of comments. Comments which run over a few lines (multiline) or single-line comments. This is an example of a multi-line comment. Everything between "/\*" and "\*/" gets ignored by the program itself. Comments are good for people who read your code, so they know what everything does so they can fix/debug it easily.

```
*****
```

```
#include <iostream.h> = This tells the compiler that the "iostream.h" library file is needed and should be included when compiling the code. A library is a collection of program code that can be included (and used) in a program to perform a variety of tasks. "iostream.h" is a library (or "header") that performs certain input and output functions. You can get these library files (if for some strange reason they didn't come with your compiler) at loads of different websites like http://code.box.sk.
```

```
*****
```

```
int main() = Okay, this is fairly complicated but bare with me. "int main()" is a function header. "main()" is the function name. A function is a collection of statements. A function header includes the return type of the function (what it returns to the place it was called when it exits) and the function name, in this case "main". This function returns an INTEGER through the line "return 0", but more on integers and the RETURN statement later. So all the functions that have an integer as the return type return integers. All the statements in a function are enclosed in curly brackets "{" and "}". This means that everything between the open bracket "{" and the closed bracket "}" are part of that function. I hope I didn't make that sound too complicated. You'll get it eventually, even if you don't understand it now. Note: For a function to be executed it needs to be called. You can call a function from anywhere, just by entering its name, like "askName()", except for the main() function. This is called automatically when the program is executed. This is why we don't call it. main() functions usually return an integer.
```

```
*****
```

```
cout << "Hello World" << endl; = This prints "Hello World" to the screen. "Hello World" is a string of text (in programming, there's no other kind of string, other than a text string). This line is an example of a stream. A stream is an input/output device. "cout" (pronounced "see-out") is the Console OUTPUT stream. "iostream.h" needs to be INCLUDED when we use streams. First we send "Hello World". We send it using an operator. The "<<" operator. This basically means, send the string to the cout stream. Then we also send "endl" (this is optional). endl means END Line, so if we send some sort text to cout, it will appear on a new line. All statements are ended with a semi-colon ";".
```

```
*****
```

```
// The first statement
// The second statement = These are both examples of one-line comments.
*****
return 0; = This returns the integer 0 from where it was called. In the case of
the main() function, this just means it exits fine. The main() function should al-
ways return a 0. More of the return command later.
*****
```

And that is helloworld.cpp explained. Just compile that and view it.

Variables.

Variables are pieces of information stored on the memory. There are three parts to a variable. Its name, type and value. Its name is just what we call it. Its value is the information it holds. This can be an integer, a decimal, a string...etc. It's type is defined in accordance to the information it is going to hold:

```
int = -32,768 to 32,767
long int = -2,147,483,648 to 2,147,483,647
float = 1.2 x 10^-38 to 3.4 x 10^38
double = 2.2 x 10^-308 to 1.8 x 10^308
char = 256 character values
```

Here is an example program which shows this:

```
***** numbers.cpp *****
/* To show variables in action
   Numbers program
   Written by Jethro */
#include <iostream.h>
#include <stdlib.h>

int main ()
{
    int num1;
    int num2;
    cout << "To show variables" << endl;
    cout << "Please enter a number: ";
    cin >> num1; // User input #1
    cout << "\nPlease enter another number: ";
    cin >> num2; // User input #2
    cout << num1 << " + " << num2 << " = " << num1+num2 << endl; // Add
    cout << num1 << " - " << num2 << " = " << num1-num2 << endl; // Sub-
tract
    system("PAUSE");
```

```

return 0;
}
*****

```

New stuff:

```
*****
```

`#include <stdlib.h>` = This is another example of using one of the countless header files available in C++. We need to include "stdlib.h" because we use the `SYSTEM()` function to interface with MS-DOS.

```
*****
```

`int num1;` and `int num2;` = Here we declare two integer-type variables. We call them (originally enough :) "num1" and "num2". But you can call them "drag" and "weed" if you want, it doesn't matter. Just remember that C++ is CaSe SeNSiTiVe, so num1 is not the same as Num1, nUm1 or even nuM1! We don't give them values, we can assign them values later... Anyway, as long as you know that that means they are integers, it's okay.

```
*****
```

`cin >> num1;` = "cin" (pronounced "see-in") is another example of a stream, courtesy of "iostream.h" again. This means that everything the user types, up to the carriage return (aka. "Enter"), will be included into the variable "num1". Notice the way we use ">>" to symbolise that the information is going from "cin" to "num1", not vica versa, as is in the "cout" stream. We did not use "endl" on the previous line, because we wanted the user to type the number right after "Please enter a number: ".

```
*****
```

`cout << "\nPlease enter another number: ";` = You probably wondering why I put '\n' there. Well, it's simple. \n means go to a new line. Similar to endl. The following line:

`'cout << "\n\n\nThree lines down";'` would print out the words "Three lines down", three new lines down.

```
*****
```

"num1+num2" and "num1-num2" = This is the use of operators. num1+num2 returns the value of num1 added to num2 and num1-num2 gives the value of num2 subtracted from num1. This does nothing to the actual value of the variable however.

+ = Addition

- = Subtraction

\* = Multiplication

/ = Division

```
*****
```

`system("PAUSE")` = The `system()` function, courtesy of "stdlib.h", lets you run an MS-DOS command. If you have read any of tutorials (I have written a few) on MS-DOS, you will know that this creates the prompt "Press any key to continue..." and will not continue executing the program until the user has pressed a key.

```
*****
```

And that's numbers.cpp...

I think that's enough for Chapter One. In Chapter Two I will talk about functions, conditional statements, loops and whatever else I think of.

If Chapter Two hasn't been written yet, check back in a few days.

\*\*\*\*\* EOF \*\*\*\*\*

Creative type of person?  
We're looking for  
cryptographic puzzles,  
challenges and other  
security type fun.  
Crosswords, word finds, etc  
with a security bent are  
welcomed. Submit to  
[mismittens@mismittens.com](mailto:mismittens@mismittens.com)  
by Friday September 20,  
2002.

# C++ Programming (Chapter Two)

## by Jethro

**Topic: Programming**

<http://jethrojones.hyperlinx.cz>

-----

This is the second chapter in my current series of Computer Programming with C++. If you haven't read Chapter One, then read it now. It might shed some light on some of the techniques used in the section. In this section we are going to cover:

Index:

- o Conditional Statements (IF...ELSE)
- o Loops (FOR...WHILE)

Here we go!

```
***** checkage.cpp *****  
  
/* This program checks  
   what age you are! !!!WOW!!!  
   Written by: Jethro */  
  
#include <iostream.h>  
  
void main() {  
  
    int age;  
    cout << "The amazing age checking program" << endl;  
    cout << "Enter you age in here please: ";  
    cin >> age;  
  
    if (age==18) {  
        cout << "\tYou are just old enough to vote" << endl;  
    }  
  
    if (age==16) {  
        cout << "\tYou are just old enough to drink" << endl;  
    }  
  
    if (age>10) {  
        cout << "\tYou are older than 10" << endl;  
    }  
}
```

```
if (age<70) {
    cout << "\tYou are younger than 70" << endl;
}

if (age!=13) {
    cout << "\tYou are not 13" << endl;
}

else {
    cout << "\tYou have some kind of crazy age!" << endl;
}

}
```

\*\*\*\*\*

Ahh, let's check this stuff out.

The conditional statement is easy to understand. If you have ever learnt a programming language in your life, then you will probably know that nearly every language has some sort of a conditional statement. A lot of languages (eg. PHP, JavaScript...etc) have this exact kind of conditional statement:

```
IF (conditional) {
statements
}
```

The "if" part is pretty easy to understand. If the (conditional) is true then the "statements" should be executed. The condition is usually in the following form:

```
IF (variable==value) {
do something
}
```

For example. In the checkage.cpp we used many variations of this. The first was:

```
if (age==18) {
    cout << "\tYou are just old enough to vote" << endl;
}
```

This means that if the age variable is equal to 18 the execute the statement:

```
'cout << "\tYou are just old enough to vote" << endl;'
```

I think that's fairly easy to understand. But what if you want to check

if it isn't equal to it, or it's bigger than that. Then you use a different operator. Here is a list of some of the most common:

\*\*\*\*\*

### Operator - Definition

-----

== - Equal To

!= - Not equal to

< - Smaller than

> - Bigger than

<= - Smaller or equal to

>= - Bigger or equal to

\*\*\*\*\*

By putting an exclamation mark (!) in front of any of these, turns them around. For example `age!<6` would be true if "age" wasn't smaller than 6. But why say this when you can say `>?`

When all else fails, use "else". This is used when none of the conditional statements are true. Of course, the else statement doesn't need a condition.

\*\*\*\*\* loops.cpp \*\*\*\*\*

```
/* This program shows
   examples of the FOR
   and WHILE loops
   Written by: Jethro */
```

```
#include <iostream.h>
```

```
void main() {
```

```
    int i, num1;
```

```
    for (i=0; i<=10; i++) {
```

```
        cout << "Saying this for time number: " << i << endl;
```

```
    }
```

```
    while (num1!=5) {
```

```
        cout << "Please enter the number 5: ";
```

```
        cin >> num1;
```

```
    }
```

```
    cout << "You got it right!" << endl;
```

```
}
```

\*\*\*\*\*

More stuff to discuss:

## FOR

---

The loop is really easy as well.

The FOR statement has a few different things packed into the one statement, but it's easy once you get the hang of it.

Let's examine/dissect it piece by piece:

```
for (i=0; i<=10; i++) {  
    cout << "Saying this for time number: " << i << endl;  
}
```

FOR - This starts off the statement

i=0 - This defines the "i" variable as being 0

i<=10 - This makes it so that "i" shouldn't go past 10

i++ - This increments the "i" variable. Basically this means "i=i+1".

"i--" is the same as "i=i-1". We add this so that every time the loop executes, "i" gets a little bit bigger. If this didn't happen, the loop would execute infinitely!

\*\*\*\*\*

Note: If you have read other tutorials on programming with loops you probably have seen them use a variable called "i" as well. This is because using "i" is an age-old tradition in loops. I don't really know the actual reason for this, but I use the i variable because it's the first one that comes to mind. You don't have to, of course.

\*\*\*\*\*

With the above loop, the loop will start at 0 and keep executing the "cout << ...etc" statement until it reaches 10.

## WHILE

-----

The WHILE statement is considerably easy to understand than FOR is. Basically, the WHILE loop will keep executing while the conditional statement is met. Kind of like a power-crazy "IF" statement. Here is the WHILE syntax:

\*\*\*\*\*

```
WHILE (condition) {  
    statement  
}
```

\*\*\*\*\*

For example:

\*\*\*\*\*

```
int jk = 0;
WHILE (jk!=5) {
cout << "\nJK is 5! Change it: ";
cin >> jk;
}
*****
```

The above loop would keep executing WHILE "jk" is not equal to 5. Of course if you use this command and don't give the program the opportunity to change it, it will just keep repeating forever and ever and ever.

And that's Chapter Two finished. Pretty easy stuff, isn't it? Chapter Three will be out soon!

\*\*\*\*\* EOF \*\*\*\*\*

We are looking for articles on intrusion detection (e.g., how snort works, NFR works, etc.), penetration auditing (e.g., method, theory, tools), forensics or other security oriented articles. Please ensure submissions are in by September 20, 2002.

# Beginners Guide To PHP

## by Rewandythal

### ***Introduction***

This guide is intended as a quick reference for beginners in PHP. It will tell you the basics you need for static PHP pages, and will include a script illustrating database connectivity and printing values from a database.

### ***What Is PHP?***

PHP is a server-side scripting language for web pages. The PHP program (or PHP web server module, depending on how PHP was compiled on your server) processes any pages requested from that server that have a .php (or .php3 for PHP3) file extension. The PHP program executes any PHP scripting commands within the file, and sends out a plain HTML file to the browser, with any additional statements added due to execution of the PHP code.

### ***Creating A PHP File***

PHP files take on the form of standard HTML files, i.e. you should use:

---

```
<html>
<head><title>Page Title</title>
</head>
<body>
...
</body>
</html>
```

---

The file should be saved with the extension '.php' to ensure that it is processed by PHP, and any PHP code should be surrounded by PHP start and end tags.

These tags exist in several forms, the first, and most common, being (XML Style)

---

```
<?php
//php code
?>
```

---

The `<?php` and `?>` signify the start and end of the PHP code.

Method 2 is a shorter version of the above, preferred by advanced web designers because it saves time and improves readability of code (SGML Style):

---

```
<?  
//php code  
?>
```

---

The third method is ASP Style, uses the following:

---

```
<%  
//php code  
% >
```

---

And finally, the fourth, Script Style, looks like this:

---

```
<script language="php">  
//php code  
</script>
```

---

Now that you know how to tell PHP that it is the start and end of PHP code, you need to know some PHP code to put between those tags!

Whilst there are a whole range of PHP functions, statements and other stuff, I am not going to go into all of them here. This is a beginners tutorial, for a complete reference, in an easy to understand format, go buy the O'Reilly Programming PHP book (ISBN 1-56592-610-2, US \$39.95, Canada \$61.95, UK £28.50).

Output from PHP can be achieved through a variety of statements, the simplest being echo.

The statement takes on the format:

```
echo "[string | $variable]"
```

You can use echo to print text out to the browser. You can echo the values of variables, by including the \$variable, and you can echo HTML statements, which the browser will then parse as usual. PHP scripts usually, but not always, print output to the browser using either echo or one of the other print statements. These include print() and printf(). Those of you familiar with C may be able to guess the syntax of the printf() function.

Variables in PHP are always prefixed with a \$ sign. They must begin with a letter, and are case-sensitive. Variables can store data of any type and PHP will convert between types should you perform operations involving more than one type. (e.g. adding a string to an integer, if the string contains only numeric values, it is converted to an integer and added. If the string contains non-numeric characters, the integer is converted to a string [i.e. 30 would become "30"] and added to the string at the appropriate place.)

Another useful PHP function is the mail() function. This allows PHP to send e-mail.

Its syntax is:

---

```
mail( string recipient, string subject, string message[, string headers[, string parameters]])
```

---

To send e-mail to myself, on my own web server, I would use the following statement:

---

```
Mail("andrew@localhost.localdomain", "PHP Form Results", $formvariables);
```

---

Remember that each statement in a PHP script must end with a semicolon, ;

## Example Scripts

These scripts were written for functionality only, and are not meant to be used "as-is" in a secured environment. Secure PHP scripting has been covered in a tutorial on AntiOnline by chsh – Go search for it.

### Example 1 – Submitting A Link To A Database

HTML File:

---

```
<html>
```

```
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
```

```
<title>Firestorm UK</title>

<link rel="stylesheet" type="text/css" href="intranet.css">

<base target="_top">

</head>

<body>

<p align="center"></p>

<p align=center><font size=5>Links</font></p>

<?php include('linkfill.inc'); ?>

<p align=center><font size=5>Submit A Link</font></p>

<form method="POST" action="linkssubmit.php">

  <p align=center>Site Name: <input type="text" name="name" size="30"></p>

  <p align=center>Link: <input type="text" name="link" size="40"></p>

  <p align=center>Description:</p>

  <p align=center><textarea rows="4" name="description" cols="40"></textarea></p>

  <p align="center"><input type="submit" value="Submit" name="B1"> <input type="reset" value="Reset" name="B2"></p>

</form>

</body>

</html>
```

---

PHP Script:

```
<html><head><title>Thankyou</title></head>

<link rel=stylesheet url="intranet.css" type="text/css">

<body>

<?php

//linkssubmit.php
```

```

require_once('DB.php');

$db = DB::connect("pgsql://postgres@localhost/firestorm");

if (DB::isError($db) {
    die($db->getMessage());
}

$name = $_POST['name'];
$link = $_POST['link'];
$description = $_POST['description'];

if(substr($link,0,7) != "http://"){
    $link = "http://" . $link;
}

$sql = "INSERT INTO links VALUES('$name','$link','$description')";
$q = $db->query($sql);
?>

Thankyou for submitting a link.<p>

Click <a href="links.php">HERE</a> to return.

</body>

</html>

```

---

SQL Table Creation:

```
CREATE TABLE links (name text,url text,description text);
```

---

## Example 2 – Getting Values From A Database & Printing To A Table

PHP Script

```

<table border=1>

<tr><th>Site</th><th>Link</th><th>Description</th></tr>

<?php

//linkfill.inc

require_once('DB.php');

$db = DB::connect("pgsql://postgres@localhost/firestorm");

```

```
if (DB::iserror($db)) {  
    die($db->getMessage());  
}  
  
//issue the query  
$sql = "SELECT links.name,links.url,links.description FROM links ORDER BY links.name ASC";  
$q = $db->query($sql);  
if (DB::iserror($q)) {  
    die($q->getMessage());  
}  
  
//generate the table  
while ($q->fetchInto($row)) {  
    ?>  
    <tr><td><?= $row[0] ?></td>  
    <td><a href="<?= $row[1] ?>"><?= $row[1] ?></a></td>  
    <td><?= $row[2] ?></td></tr>  
    <?php  
    }  
    ?>  
    </table>
```

---

### Example 3 – Validating User Login

Very insecure, but allows multiple users in an environment where security is not a concern (e.g. internally on a home network not connected to the internet)

HTML File:

```
<html><head><title>Archives</title></head>  
<link rel=stylesheet href="intranet.css" type="text/css">  
<body>  
<p align=center></p>
```

```
<table align="center" border="0" width="50%">
<tr><td>
<p align="center"><b>Please login for your personal archives</b></p>
<form action="archivelogin.php" method="POST">
<p align="center">Username: <input type="text" name="username" size="10"><br>
Password: <input type="password" name="password" size="10"></p>
<p align="center"><input type="submit" name="submit" value="Login"> <input type="reset" name="reset" value="Clear"></p>
</form></td></tr>
</table>
</body>
</html>
```

---

PHP File:

---

```
<head><base target="_top"></head><?php
$username = $_POST['username'];
$password = $_POST['password'];
require_once('DB.php');
$db = DB::connect("pgsql://postgres@localhost/firestorm");
if (DB::iserror($db))
{
    die($db->getMessage());
}
$sql = "SELECT users.username,users.password,users.css FROM users ORDER BY users.username ASC";
$q = $db->query($sql);
if (DB::iserror($q))
{
    die($q->getMessage());
}
while ($q->fetchInto($row))
{
```

```
if ( $username == $row[0])
{
    if ( $password == $row[1])
    {
        $css = $row[2];
        include "archive.inc";
    }
}
}
if ($q == false)
{
    ?>
    <h2>Error: Invalid Login</h2>
    <?
}
?>
```

---

PHP Included File:

---

```
<html><head><title>Personal Archive</title>

</head>
<link rel=stylesheet href="<?= $css ?>" type="text/css">
<body>
<p align=center></p>
<table border=0>
<?php
// archive.inc
// User dependent archive-filler
// Takes data from "archives" table based on username
if ($username == NULL)
{
```

```
        die("Invalid Login.");
    }
require_once('DB.php');
$db = DB::connect("pgsql://postgres@localhost/firestorm");
if (DB::iserror($db))
{
    die($db->getMessage());
}
$sql = "SELECT content FROM archives WHERE username='$username'";
$q = $db->query($sql);
if (DB::iserror($q))
{
    die($q->getMessage());
}
while ($q->fetchInto($row))
{
    ?>
    <?= $row[0] ?>
    <?php
}
?>
```

---

SQL Table Creation:

---

```
CREATE TABLE users (username text,password text,css text);
```

---

The above scripts validate a login and set a personal CSS (Stylesheet) for each user, depending on their database entry.