PHP

PROF. ROHINI D. DHAGE

Introduction:

1.1 What Does PHP Do?

PHP can be used in three primary ways:

- **Server-side scripting**

  PHP was originally designed to create dynamic web content, and it is still best suited for that task. To generate HTML, you need the PHP parser and a web server to send the documents. PHP has also become popular for generating XML documents, graphics, Flash animations, PDF files, and more.

- **Command-line**

  Scripting PHP can run scripts from the command line, much like Perl, awk, or the Unix shell. You might use the command-line scripts for system administration tasks, such as backup and log parsing.

- **Client-side GUI applications**

  Using PHP-GTK (http://gtk.php.net), you can write full-blown, cross-platform GUI applications in PHP.

1.2. A Brief History of PHP :

Rasmus Lerdorf first conceived of PHP in 1994, but the PHP that people use today is quite different from the initial version. To understand how PHP got where it is today, it is useful to know the historical evolution of the language. Here's that story, as told by Rasmus.

1.2.1. The Evolution of PHP :

Here is the PHP 1.0 announcement that I posted to the Usenet

News group comp.Infosystems.www.authoring.cgi in June 1995:

The only requirement for these tools to work is that you have

Over the next year or so, a lot was done and the focus changed quite a bit. Here's the PHP Version 2(PHP/FI) announcement I sent in April 1996.

This was the first time the term "scripting language" was used. PHP 1's simplistic tag-replacement code was replaced with a parser that could handle a more sophisticated embedded tag language. By today's standards, the tag language wasn't particularly sophisticated, but compared to PHP 1 it certainly was.

The main reason for this change was that few people who used PHP 1 were actually interested in using the C-based framework for creating add-ons. Most users were much more interested in being able to embed logic directly in their web pages for creating conditional HTML, custom tags, and other such features. PHP 1 users were constantly requesting the ability to add the hit-tracking footer or send different HTML blocks conditionally. This led to the creation of an `if` tag. Once you have `if` , you need `else` as well and from there, it's a slippery slope to the point where, whether you want to or not, you end up writing an entire scripting language.

By mid-1997, PHP Version 2 had grown quite a bit and had attracted a lot of users, but there were still some stability problems with the underlying parsing engine. The project was also still mostly a one-man effort, with a few contributions here and there. At this point, Zeev Suraski and Andi Gutmans in Tel Aviv volunteered to rewrite the underlying parsing engine, and we agreed to make their rewrite the base for PHP Version 3. Other people also volunteered to work on other parts of PHP, and the project changed from a one-person effort with a few contributors to a true open source project with many developers around the world.

Here is the PHP 3.0 announcement from June 1998:
After the release of PHP 3, usage really started to take off. Version 4 was prompted by a number of developers who were interested in making some fundamental changes to the architecture of PHP. These changes included abstracting the layer between the language and the web server, adding a thread-safety mechanism, and adding a more advanced, two-stage parse/execute tag-parsing system. This new parser, primarily written by Zeev and Andi, was named the Zend engine. After a lot of work by a lot of developers, PHP 4.0 was released on May 22, 2000.

Since that release, there have been a few minor releases of PHP 4, with the latest version as of this writing being 4.3.11. , PHP Version 5 has been released for some time. There have already been a few minor 'dot' releases, and the stability of this current version is quite high. As you will see, there have been some major advances made in this version of PHP.XML, object orientation, and SQLite are among the major updates. Many other minor changes, function additions, and feature enhancements have also been incurporated.

1.2.2. The Growth of PHP :

Figure shows the growth of PHP as measured by the usage numbers collected by Net craft(http://www.netcraft.com) since January 2000. This figure shows the total number of unique IP addresses that report they are using Apache with the PHP module enabled (PHP: 19,720,597Domains, 1,310,181 IP Addresses). The slight dip at the end of 2001 reflects the demise of a number of dot-coms that disappeared from the Web. The overall number of servers that Net craft found also went down for the first time during this period. You can see an update of this chart for yourself at anytime by accessing this web address: http://www.php.net/usage.php

Latest version of PHP is 8.0.1 (Released Jan 7,2021)

1.3. Installing PHP :

PHP is available for many operating systems and platforms. The most common setup, however, is to use PHP as a module for the Apache web server on a Unix machine. We will briefly describes how to install Apache with PHP.

To install Apache with PHP, you'll need a Unix machine with an ANSI-compliant C compiler, and around 10 MB of available disk space for source and object files. You'll also need Internet access to fetch the source code for PHP and Apache. Start by downloading the source distributions of PHP and Apache.

The latest files are always available from the web sites for the respective tools. Since there are so many options on installation , we are showing here the generic installation instructions for a Linux server as shown on the PHP web site athttp://ca3.php.net/manual/en/install.unix.php. You will have to replace the `xxx` signifier in the following steps with the version of the software that you choose to install.

2.1. Lexical Structure :

The lexical structure of a programming language is the set of basic rules that governs how you write programs in that language. It is the lowest-level syntax of the language and specifies such things as what variable names look like, what characters are used for comments, and how program statements are separated from each other.

2.1.1. Case Sensitivity:

The names of user-defined classes and functions, as well as built-in constructs and keywords such as echo, while, class, etc., are case-insensitive. Thus, these three lines are equivalent:

```
echo("hello, world");
ECHO("hello, world");
EcHo("hello, world");
```

Variables, on the other hand, are case-sensitive. That is, $name, $NAME, and $NaME are three different variables.

2.1.2. Statements and Semicolons:

A statement is a collection of PHP code that does something. It can be as simple as a variable assignment or as complicated as a loop with multiple exit points. Here is a small sample of PHP statements, including function calls, assignment, and an if test:

```
echo "Hello, world";
myfunc(42, "O'Reilly");
$a = 1; $name = "Elphaba";
$b = $a / 25.0;
if ($a == $b) { echo "Rhyme? And Reason?"; }
```

PHP uses semicolons to separate simple statements. A compound statement that uses curly braces tomark a block of code, such as a conditional test or loop, does not need a semicolon after a closingbrace. Unlike in other languages, in PHP the semicolon before the closing brace is not optional:

```
if ($needed) {
    echo "We must have it!";   // semicolon required here
  }                            // no semicolon required here after the brace
```

The semicolon is optional before a closing PHP tag:

```
<?php
    if ($a == $b) { echo "Rhyme? And Reason?"; }
    echo "Hello, world"                 // no semicolon required before closing tag
?>
```

It's good programming practice to include optional semicolons, as they make it easier to add codelater.

2.1.3. Whitespace and Line Breaks :

In general, whitespace doesn't matter in a PHP program. You can spread a statement across any number of lines, or lump a bunch of statements together on a single line. For example, this statement:

```
raise_prices($inventory, $inflation, $cost_of_living, $greed);
```

could just as well be written with more whitespace:

```
raise_prices (
                $inventory          ,
                $inflation          ,
                $cost_of_living   ,
                $greed ) ;
```

2.1.4. Comments :

Comments give information to people who read your code, but they are ignored by PHP. Even if you think you're the only person who will ever read your code, it's a good idea to include comments in your code in retrospect, code you wrote months ago can easily look as though a stranger wrote it.

For example, this is worthless:

```
$x = 17; // store 17 into the variable $x
```

whereas this will help whoever maintains your code:

```
// convert &#nnn; entities into characters
$text = preg_replace('/&#([0-9])+;/e', "chr('\\1')", $text);
```

PHP provides several ways to include comments within your code, all of which are borrowed from existing languages such as C, C++, and the Unix shell. In general, use C-style comments to comment out code, and C++-style comments to comment on code.

2.1.4.1. Shell-style comments :

When PHP encounters a hash mark (#) within the code, everything from the hash mark to the end of the line or the end of the section of PHP code (whichever comes first) is considered a comment. This method of commenting is found in Unix shell scripting languages and is useful for annotating single lines of code or making short notes.

Because the hash mark is visible on the page, shell-style comments are sometimes used to mark off blocks of code:

```
 #########################
## Cookie functions
#########################
```

Sometimes they're used before a line of code to identify what that code does, in which case they'reusually indented to the same level as the code:

```
If ($double_check) {
 # create an HTML form requesting that the user confirm the action
echo confirmation_form( );
 }
```

2.1.5. Literals :

A literal is a data value that appears directly in a program. The following are all literals in PHP:

```
2001
0xFE
1.4142
"Hello World" ‘
Hi‘
 true
null
```

2.1.6. Identifiers :

An identifier is simply a name. In PHP, identifiers are used to name variables, functions , constants,and classes. The first character of an identifier must be either an ASCII letter (uppercase orlowercase), the underscore character (_), or any of the characters between ASCII 0x7F and ASCII0xFF. After the initial character, these characters and the digits 0-9 are valid.

2.1.6.1. Variable names :

Variable names always begin with a dollar sign ($) and are case-sensitive. Here are some valid variable names :

```
$bill
$head_count
$MaximumForce
$I_HEART_PHP
$_underscore
$_int
```

Here are some illegal variable names:

```
$not valid
$|
$3wa
```

These variables are all different:

```
$hot_stuff $Hot_stuff $hot_Stuff $HOT_STUFF
```

2.1.6.2. Function names :

Function names are not case-sensitive. Here aresome valid function names:

```
tally
list_all_users
deleteTclFiles
LOWERCASE_IS_FOR_WIMPS
_hide
```

These function names refer to the same function:

```
howdy        HoWdY        HOWDY    HOWdy    howdy
```

2.1.6.3. Class names :

Class names follow the standard rules for PHP identifiers and are not case-sensitive. Here are some valid class names:

```
Person
account
```

The class name std Class is reserved.

2.1.6.4. Constants :

A constant is an identifier for a simple value; only scalar values Boolean, integer, double, and string can be constants . Once set, the value of a constant cannot change. Constants are referred to by their identifiers and are set using the define( ) function:

```
define('PUBLISHER', "O'Reilly & Associates");
echo PUBLISHER;
```

2.1.7. Keywords :

A keyword (or reserved word) is a word reserved by the language for its core functionality you cannot give a variable, function, class, or constant the same name as a keyword. Table 2-1 lists the keywords in PHP, which are case-insensitive.

Table PHP core language keywords

_ _CLASS_ _	clone	endif
_ _FILE_ _	Const	endswitch
_ _FUNCTION_ _	Continue	endwhile
_ _LINE_ _	Declare	eval( )
_ _METHOD_ _	Default	exception
Abstract	die( )	exit( )
And	Do	extends
array( )	echo( )	extends
As	Else	final
Break	elseif	for
Case	empty()	foreach
Catch	enddeclare	function

2.2. Data Type

PHP provides eight types of values, or data types. Four are scalar (single-value) types: integers ,floating-point numbers, strings, and Booleans. Two are compound (collection) types: arrays and objects. The remaining two are special types: resource and NULL. Numbers, Booleans, resources, and NULL

2.2.1. Integers

Integers are whole numbers, such as 1, 12, and 256. The range of acceptable values varies according to the details of your platform but typically extends from -2,147,483,648 to +2,147,483,647.Specifically, the range is equivalent to the range of the long data type of your C compiler. Unfortunately, the C standard doesn't specify what range that long type should have, so on some systems you might see a different integer range. Integer literals can be written in decimal, octal, or hexadecimal. Decimal values are represented by a sequence of digits, without leading zeros. The sequence may begin with a plus (+) or minus (-) sign. If there is no sign, positive is assumed. Examples of decimal integers include the following:

```
1998
 -641
 +33
```

Octal numbers consist of a leading 0 and a sequence of digits from 0 to 7. Like decimal numbers, octal numbers can be prefixed with a plus or minus. Here are some example octal values and their equivalent decimal values:

```
0755          // decimal 493
+010          // decimal 8
```

Hexadecimal values begin with 0x, followed by a sequence of digits (0-9) or letters (A-F). The letters can be upper- or lowercase but are usually written in capitals. Like decimal and octal values, you can include a sign in hexadecimal numbers :

```
0xFF                // decimal
255 0x10            // decimal 16
-0xDAD1             // decimal -56017
```

If you try to store a variable that is too large to be stored as an integer, or is not a whole number, it will automatically be turned into a floating-point number.

Use the is_int( ) function (or its is_integer( ) alias) to test whether a value is an integer:

```
if (is_int($x)) {
        // $x is an integer
}
```

2.2.2. Floating-Point Numbers :

Floating-point numbers (often referred to as real numbers) represent numeric values with decimal digits. Like integers, their limits depend on your machine's details. PHP floating-point numbers are equivalent to the range of the double data type of your C compiler. Usually, this allows numbers between 1.7E-308 and 1.7E+308 with 15 digits of accuracy. If you need more accuracy or a wider range of integer values, you can use the BC or GMP extensions. See Appendix B for an overview of the BC and GMP extensions.

PHP recognizes floating-point numbers written in two different formats. There's the one we all use every day:

```
3.14
0.017
-7.1
```

but PHP also recognizes numbers in scientific notation:

```
0.314E1          // 0.314*101, or   3.14
17.0E-3          // 17.0*10-3, or   0.017
```

Floating-point values are only approximate representations of numbers. For example, on many systems 3.5 is actually represented as 3.4999999999. This means you must take care to avoid writing code that assumes floating-point numbers are represented completely accurately, such as directly comparing two floating-point values using ==. The normal approach is to compare to several decimal places:

2.2.3. Strings :

Because strings are so common in web applications, PHP includes core-level support for creating and manipulating strings. A string is a sequence of characters of arbitrary length. String literals are delimited by either single or double quotes. But both are treated differently.
A string is a non-numeric data type. It holds letters or any alphabets, numbers, and even special characters.
But both are treated differently. To clarify this, see the example below:

Example:

```
'big dog’
"fat hog“
```

Variables are expanded within double quotes, while within single quotes they are not:

```
$name = "Guido";
echo "Hi, $name\n";
echo 'Hi, $name’;
Hi, Guido
Hi, $name
```

```
<?php
        $company = “PHP";
        //both single and double quote statements will treat different
        echo "Hello $company";
        echo "</br>";
        echo 'Hello $company’;
?>
```

Output:

```
Hello PHP Hello $company
```

2.2.4. Booleans :

Booleans are the simplest data type works like switch. It holds only two values: **TRUE (1)** or **FALSE (0)**. It is often used with conditional statements. If the condition is correct, it returns TRUE otherwise FALSE.

Example:

```
<?php
    if (TRUE)
            echo "This condition is TRUE.";
.   if (FALSE)
             echo "This condition is FALSE.";
?>
```

Output:
This condition is TRUE

2.2.5 Array :

An array is a compound data type. It can store multiple values of same data type in a single variable.

Example:

```
<?php
    $bikes = array ("Royal Enfield", "Yamaha", "KTM");
    var_dump($bikes);   //the var_dump() function returns the datatype and values
    echo "</br>";
    echo "Array Element1: $bikes[0] </br>";
    echo "Array Element2: $bikes[1] </br>";
    echo "Array Element3: $bikes[2] </br>";
?>
```

Output:

```
array(3) { [0]=> string(13) "Royal Enfield" [1]=> string(6) "Yamaha" [2]=> string(3) "KTM" }Array
Element1: Royal Enfield
Element2: Yamaha
Element3: KTM
```

2.2.6 PHP object :

Objects are the instances of user-defined classes that can store both values and functions. They must be explicitly declared.

Example:

```
<?php
   class bike
  {
       function model()
      {
              $model_name = "Royal Enfield";
              echo "Bike Model: " .$model_name;
       }
  }
  $obj = new bike();
  $obj -> model();
?>
```

Output:

```
Bike Model: Royal Enfield
```

2.2.7 PHP Resource :

Many modules provide several functions for dealing with the outside world. For example, every database extension has at least a function to connect to the database, a function to send a query to the database, and a function to close the connection to the database. Because you can have multiple database connections open at once, the connect function gives you something by which to identify hat connection when you call the query and close functions: a resource (or a "handle").
Resources are not the exact data type in PHP. Basically, these are used to store some function calls or references to external PHP resources.
For example - a database call. It is an external resource. This is an advanced topic of PHP.

2.2.8 PHP Null :

Null is a special data type that has only one value: **NULL**. There is a convention of writing it in capital letters as it is case sensitive.
The special type of data type NULL defined a variable with no value.

Example:

```
<?php
    $nl = NULL;
    echo $nl;   //it will not give any output
?>
```

PHP Variables :

In PHP, a variable is declared using a **$ sign** followed by the variable name. Here, some important points to know about variables:

- As PHP is a loosely typed language, so we do not need to declare the data types of the variables. It automatically analyses the values and makes conversions to its correct datatype.
- After declaring a variable, it can be reused throughout the code.
- Assignment Operator (=) is used to assign the value to a variable.

Syntax of declaring a variable in PHP is given below:

```
$variablename=value;
```

Rules for declaring PHP variable:

- A variable must start with a dollar ($) sign, followed by the variable name.
- It can only contain alpha-numeric character and underscore (A-z, 0-9, _).
- A variable name must start with a letter or underscore (_) character.
- A PHP variable name cannot contain spaces.
- One thing to be kept in mind that the variable name cannot start with a number or special symbols.
- PHP variables are case-sensitive, so $name and $NAME both are treated as different variable.

PHP Variable: Declaring string, integer, and float :

Let's see the example to store string, integer, and float values in PHP variables.

```
<?php
	$str="hello string";
	$x=200;
	$y=44.6;
	echo "string is: $str <br/>";
	echo "integer is: $x <br/>";
	echo "float is: $y <br/>";
?>
```

Output:

```
string is: hello string
integer is: 200
float is: 44.6
```

PHP Variable: Sum of two variables :

```
<?php
        $x=5;
        $y=6;
        $z=$x+$y;
        echo $z;
?>
```

Output:

```
11
```

PHP Variable: case sensitive :

In PHP, variable names are case sensitive. So variable name "color" is different from Color, COLOR, COLor etc.

```
<?php
$color="red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>";
echo "My boat is " . $coLOR . "<br>";
?>
```

Output:

```
My car is red
Notice: Undefined variable: COLOR in C:\wamp\www\variable.php on line 4
My house is
Notice: Undefined variable: coLOR in C:\wamp\www\variable.php on line 5
My boat is
```

PHP Variable: Rules :

PHP variables must start with letter or underscore only.

PHP variable can't be start with numbers and special symbols.

```
<?php

        $a="hello";//letter (valid)

        $_b="hello";//underscore (valid)

        echo "$a <br/> $_b";
?>
```

Output:

```
hello

hello
```

```
<?php
        $4c="hello";//number (invalid)

        $*d="hello";//special symbol (invalid)

        echo "$4c <br/> $*d";

?>
```

Output:

```
Parse error: syntax error, unexpected '4' (T_LNUMBER), expecting variable (T_VARIABLE)

 or '$' in C:\wamp\www\variableinvalid.php on line 2
```

PHP Variable Scope :

The scope of a variable is defined as its range in the program under which it can be accessed. In other words, "The scope of a variable is the portion of the program within which it is defined and can be accessed.“

PHP has three types of variable scopes:

1. Local variable
2. Global variable
3. Static variable

Local variable :

The variables that are declared within a function are called local variables for that function. These local variables have their scope only in that particular function in which they are declared. This means that these variables cannot be accessed outside the function, as they have local scope.
A variable declaration outside the function with the same name is completely different from the variable declared inside the function. Let's understand the local variables with the help of an example:

```
<?php
    function local_var()
    {
        $num = 45;  //local variable
        echo "Local variable declared inside the function is: ". $num;
    }
    local_var();
?>
```

Output:

```
Local variable declared inside the function is: 45
```

```
<?php
   function mytest()
   {
         $lang = "PHP";
      echo "Web development language: " .$lang;
   }
   mytest();
   //using $lang (local variable) outside the function will generate an error
   echo $lang;
?>
```

Output:

```
Web development language: PHP
Notice: Undefined variable: lang in D:\xampp\htdocs\program\p3.php on line 7
```

Global variable :

The global variables are the variables that are declared outside the function. These variables can be accessed anywhere in the program. To access the global variable within a function, use the GLOBAL keyword before the variable. However, these variables can be directly accessed or used outside the function without any keyword. Therefore there is no need to use any keyword to access a global variable outside the function.

Let's understand the global variables with the help of an example:

Example:

```
<?php
 $name = “Rohini Dhage";        //Global Variable
  function global_var()
  {
       global $name;
       echo "Variable inside the function: ". $name;
       echo "</br>";
   }
   global_var();
   echo "Variable outside the function: ". $name;
?>
```

Output:

```
Variable inside the function: Rohini Dhage
Variable outside the function: Rohini Dhage
```

Static variable :

It is a feature of PHP to delete the variable, once it completes its execution and memory is freed. Sometimes we need to store a variable even after completion of function execution. Therefore, another important feature of variable scoping is static variable. We use the static keyword before the variable to define a variable, and this variable is called as **static variable**.
Static variables exist only in a local function, but it does not free its memory after the program execution leaves the scope. Understand it with the help of an example:

Example:

```
<?php
   function static_var()
   {
      static $num1 = 3;       //static variable
      $num2 = 6;          //Non-static variable
      //increment in non-static variable
      $num1++;
      //increment in static variable
      $num2++;
      echo "Static: " .$num1 ."</br>";
      echo "Non-static: " .$num2 ."</br>";
   }

//first function call
   static_var();

   //second function call
   static_var();
?>
```

Output:

```
Static: 4
Non-static: 7
Static: 5
Non-static: 7
```

You have to notice that $num1 regularly increments after each function call, whereas $num2 does not. This is why because $num1 is not a static variable, so it freed its memory after the execution of each function call.

Garbage Collection :

PHP uses reference counting and copy-on-write to manage memory. Copy-on-write ensures that memory isn't wasted when you copy values between variables, and reference counting ensures that memory is returned to the operating system when it is no longer needed.

To understand memory management in PHP, you must first understand the idea of a symbol table. There are two parts to a variable its name (e.g., $name), and its value (e.g., "Fred"). A symbol table is an array that maps variable names to the positions of their values in memory.

When you copy a value from one variable to another, PHP doesn't get more memory for a copy of the value. Instead, it updates the symbol table to say "both of these variables are names for the same chunk of memory." So the following code doesn't actually create a new array:

```
$worker = array("Fred", 35, "Wilma");
$other = $worker;                    // array isn't copied
```

If you subsequently modify either copy, PHP allocates the required memory and makes the copy:

```
$worker[1] = 36; // array is copied, value changed
```

By delaying the allocation and copying, PHP saves time and memory in a lot of situations. This is copy-on-write.

Expressions and Operators :

An expression is a bit of PHP that can be evaluated to produce a value. The simplest expressions are literal values and variables. A literal value evaluates to itself, while a variable evaluates to the value stored in the variable. More complex expressions can be formed using simple expressions and operators. An operator takes some values (the operands) and does something (for instance, adds them together). Operators are written as punctuation symbols for instance, the + and - familiar to us from math. Some operators modify their operands, while most do not.

Number of Operands :

Most operators in PHP are binary operators; they combine two operands (or expressions) into a single, more complex expression. PHP also supports a number of unary operators, which convert a single expression into a more complex expression. Finally, PHP supports a single ternary operator that combines three expressions into a single expression.

Operator Precedence :

The order in which operators in an expression are evaluated depends on their relative precedence. For example, you might write:

```
2 + 4 * 3
```

The addition and multiplication operators have different precedence, with multiplication higher than addition. So the multiplication happens before the addition, giving 2 + 12,or 14, as the answer. If the precedence of addition and multiplication were reversed, 6 * 3, or 18,would be the answer. To force a particular order, you can group operands with the appropriate operator in parentheses. In our previous example, to get the value 18, you can use this expression:

```
(2 + 4) * 3
```

One way many programmers deal with the complex precedence rules in programming languages is to reduce precedence down to two rules:

- Multiplication and division have higher precedence than addition and subtraction.
- Use parentheses for anything else.

Operator Associativity :

Associativity defines the order in which operators with the same order of precedence are evaluated. For example, look at:
2 / 2 * 2
The division and multiplication operators have the same precedence, but the result of the expression depends on which operation we do first:
2/(2*2) // 0.5
(2/2)*2 // 2
The division and multiplication operators are left-associative; this means that in cases of ambiguity, the operators are evaluated from left to right. In this example, the correct result is 2.

Implicit Casting :

Many operators have expectations of their operands for instance, binary math operators typically require both operands to be of the same type. PHP's variables can store integers, floating-point numbers, strings, and more, and to keep as much of the type details away from the programmer as possible, PHP converts values from one type to another as necessary.

The conversion of a value from one type to another is called casting. This kind of implicit casting is called type juggling in PHP.

Operators :

The arithmetic operators are operators you'll recognize from everyday use. Most of the arithmetic operators are binary; however, the arithmetic negation and arithmetic assertion operators are unary. These operators require numeric values, and non-numeric values are converted into numeric values .The arithmetic operators are:

Arithemetic Operator	Assignment Operators
Bitwise Operator	Comparison Operators
Incrementing/Decrementing Operators	Logical Operators
String Operators	Array Operators
Type Operators	Execution Operators
Error Control Operators	

Arithmetic Operators :

The PHP arithmetic operators are used to perform common arithmetic operations such as addition, subtraction, etc. with numeric values.

Operator	Name	Example	Explanation
+	Addition	$a + $b	Sum of operands
-	Subtraction	$a - $b	Difference of operands
*	Multiplication	$a * $b	Product of operands
/	Division	$a / $b	Quotient of operands
%	Modulus	$a % $b	Remainder of operands
**	Exponentiation	$a ** $b	$a raised to the power $b

Assignment Operators :

The assignment operators are used to assign value to different variables. The basic assignment operator is "=".

Operator	Name	Example	Explanation
=	Assign	$a = $b	The value of right operand is assigned to the left operand.
+=	Add then Assign	$a += $b	Addition same as $a = $a + $b
-=	Subtract then Assign	$a -= $b	Subtraction same as $a = $a - $b
*=	Multiply then Assign	$a *= $b	Multiplication same as $a = $a * $b
/=	Divide then Assign (quotient)	$a /= $b	Find quotient same as $a = $a / $b
%=	Divide then Assign (remainder)	$a %= $b	Find remainder same as $a = $a % $b

Bitwise Operators :

The bitwise operators are used to perform bit-level operations on operands. These operators allow the evaluation and manipulation of specific bits withi
the integer.

Operator	Name	Example	Explanation
&	And	$a & $b	Bits that are 1 in both $a and $b are set to 1, otherwise 0.
|	Or (Inclusive or)	$a | $b	Bits that are 1 in either $a or $b are set to 1
^	Xor (Exclusive or)	$a ^ $b	Bits that are 1 in either $a or $b are set to 0.
~	Not	~$a	Bits that are 1 set to 0 and bits that are 0 are set to 1
<<	Shift left	$a << $b	Left shift the bits of operand $a $b steps
>>	Shift right	$a >> $b	Right shift the bits of $a operand by $b number of places

Comparison Operators :

Comparison operators allow comparing two values, such as number or string. Below the list of comparison operators are given.

Operator	Name	Example	Explanation
==	Equal	$a == $b	Return TRUE if $a is equal to $b
===	Identical	$a === $b	Return TRUE if $a is equal to $b, and they are of same data type
!==	Not identical	$a !== $b	Return TRUE if $a is not equal to $b, and they are not of same data type
!=	Not equal	$a != $b	Return TRUE if $a is not equal to $b
<>	Not equal	$a <> $b	Return TRUE if $a is not equal to $b
<	Less than	$a < $b	Return TRUE if $a is less than $b
>	Greater than	$a > $b	Return TRUE if $a is greater than $b
<=	Less than or equal to	$a <= $b	Return TRUE if $a is less than or equal $b
>=	Greater than or equal to	$a >= $b	Return TRUE if $a is greater than or equal $b
<=>	Spaceship	$a <=>$b	Return -1 if $a is less than $b Return 0 if $a is equal $b Return 1 if $a is greater than $b

Incrementing/Decrementing Operators :

The increment and decrement operators are used to increase and decrease the value of a variable.

Operator	Name	Example	Explanation
++	Increment	++$a	Increment the value of $a by one, then return $a
		$a++	Return $a, then increment the value of $a by one
--	decrement	--$a	Decrement the value of $a by one, then return $a
		$a--	Return $a, then decrement the value of $a by one

Logical Operators :

The logical operators are used to perform bit-level operations on operands. These operators allow the evaluation and manipulation of specific bits within the integer.

Operator	Name	Example	Explanation
and	And	$a and $b	Return TRUE if both $a and $b are true
Or	Or	$a or $b	Return TRUE if either $a or $b is true
xor	Xor	$a xor $b	Return TRUE if either $ or $b is true but not both
!	Not	! $a	Return TRUE if $a is not true
&&	And	$a && $b	Return TRUE if either $a and $b are true
||	Or	$a || $b	Return TRUE if either $a or $b is true

String Operators :

The string operators are used to perform the operation on strings. There are two string operators in PHP, which are given below:

Operator	Name	Example	Explanation
.	Concatenation	$a . $b	Concatenate both $a and $b
.=	Concatenation and Assignment	$a .= $b	First concatenate $a and $b, then assign the concatenated string to $a, e.g. $a = $a . $b

Array Operators :

The array operators are used in case of array. Basically, these operators are used to compare the values of arrays.

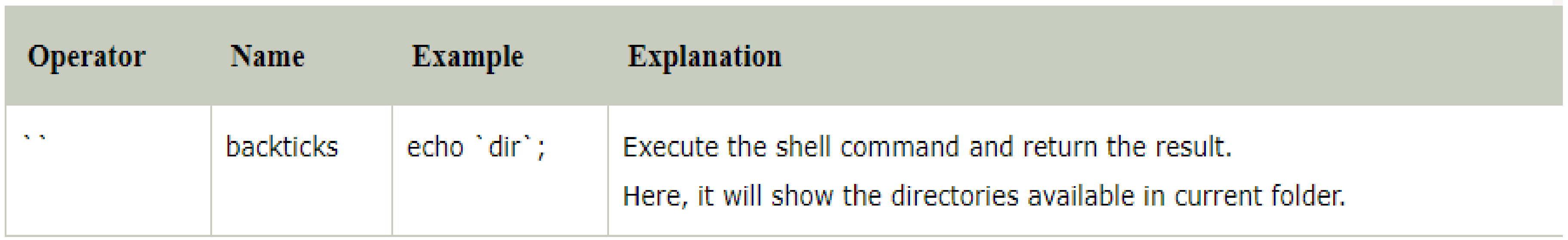
Operator	Name	Example	Explanation
+	Union	$a + $y	Union of $a and $b
==	Equality	$a == $b	Return TRUE if $a and $b have same key/value pair
!=	Inequality	$a != $b	Return TRUE if $a is not equal to $b
===	Identity	$a === $b	Return TRUE if $a and $b have same key/value pair of same type in same order
!==	Non-Identity	$a !== $b	Return TRUE if $a is not identical to $b
<>	Inequality	$a <> $b	Return TRUE if $a is not equal to $b

Type Operators :

The type operator **instanceof** is used to determine whether an object, its parent and its derived class are the same type or not. Basically, this operator determines which certain class the object belongs to. It is used in object-oriented programming.

Execution Operators :

PHP has an execution operator **backticks (``)**. PHP executes the content of backticks as a shell command. Execution operator and **shell_exec()** give the same result.



Operator	Name	Example	Explanation
` `	backticks	echo `dir`;	Execute the shell command and return the result. Here, it will show the directories available in current folder.

Error Control Operators :

PHP has one error control operator, i.e., **at (@) symbol**. Whenever it is used with an expression, any error message will be ignored that might be generated by that expression.

Operator	Name	Example	Explanation
@	at	@file ('non_existent_file')	Intentional file error

Flow-Control Statements :

PHP supports a number of traditional programming constructs for controlling the flow of execution of a program. Conditional statements, such as if/else and switch, allow a program to execute different pieces of code, or none at all, depending on some condition. Loops, such as while and for, support the repeated execution of particular segments of code.

If Statement :

The if statement checks the truthfulness of an expression and, if the expression is true, evaluates a statement. An if statement looks like:

Syntax :

```
if (expression)
        statement
```

To specify an alternative statement to execute when the expression is false, use the else keyword:

```
if (expression)
```

if...else Statement :

The if...else statement executes some code if a condition is true and another code if that condition is false.

Syntax :

```
if (condition)
{
        code to be executed if condition is true;
} else
{
 code to be executed if condition is false;
}
```

Switch Statement :

Use the switch statement to **select one of many blocks of code to be executed**.

Syntax

```
switch (n)
 {
case label1:
                code to be executed if n=label1;
                break;
case label2:
                code to be executed if n=label2;
                break;
case label3:
                code to be executed if n=label3;
                break;
                 ...
default:
                code to be executed if n is different from all labels;
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared wit
for each case in the structure. If there is a match, the block of code associated with that case is executed. Use break to prevent the code from running int
automatically. The default statement is used if no match is found.

PHP Loops :

Often when you write code, you want the same block of code to run over and over again a certain number of times. So, instead of adding several almost equal code-lines in a script, we can use loops.
Loops are used to execute the same block of code again and again, as long as a certain condition is true.
In PHP, we have the following loop types:

- while - loops through a block of code as long as the specified condition is true
- do...while - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- for - loops through a block of code a specified number of times
- foreach - loops through a block of code for each element in an array

while Loop :

The while loop executes a block of code as long as the specified condition is true.

Syntax

```
while (condition is true) {
        code to be executed;
}
```

do...while Loop :

The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

Syntax

```
do {
        code to be executed;
} while (condition is true);
```

for Loop :

The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (init counter; test counter; increment counter) {
        code to be executed for each iteration;
}
```

Parameters:

init counter: Initialize the loop counter value
test counter: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
increment counter: Increases the loop counter value

For each Loop :

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

Syntax

```
foreach ($array as $value) {
  code to be executed;
}
```

For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

PHP Break :

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch statement.
The break statement can also be used to jump out of a loop.

PHP Continue :

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

Embedding PHP in Web Pages :

Although it is possible to write and run stand alone PHP programs, most PHP code is embedded in HTML or XML files. This is, after all, why it was created in the first place. Processing such documents involves replacing each chunk of PHP source code with the output it produces when executed.

Because a single file contains PHP and non-PHP source code, we need a way to identify the regions of PHP code to be executed. PHP provides four different ways to do this.

As you'll see, the first, and preferred, method looks like XML. The second method looks like SGML. The third method is based on ASP tags. The fourth method uses the standard HTML <script> tag; this makes it easy to edit pages with enabled PHP using a regular HTML editor.

XML Style :

Because of the advent of the eXtensible Markup Language (XML) and the migration of HTML to an XML language (XHTML), the currently preferred technique for embedding PHP uses XML-compliant tags to denote PHP instructions.

Coming up with tags to demark PHP commands in XML was easy, because XML allows the definition of new tags. To use this style, surround your PHP code with <?php and ?>. Everything between these markers is interpreted as PHP, and everything outside the markers is not. Although it is not necessary to include spaces between the markers and the enclosed text, doing so improves readability. For example, to get PHP to print "Hello, world", you can insert the following line in a webpage:

```
<?php echo "Hello, world"; ?>
```

The trailing semicolon on the statement is optional, because the end of the block also forces the end of the expression. Embedded in a complete HTML file, this looks like:

```
<!doctype html public "-//w3c//DTD XHTML 1.0 Transitional//EN" http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
 <html>
        <head>
                        <title>This is my first PHP program!</title>
        </head>
<body>
        <p>
                        Look, ma! It's my first PHP program:<br />
                        <?php echo "Hello, world"; ?><br />
                        How cool is that?
        </p>
</body>
</html>
```

Also notice that we switched between PHP and non-PHP, all in the space of a single line. PHP instructions can be put anywhere in a file, even within valid HTML tags. For example:

```
<input type="text" name="first_name"
value="<?php echo "Rasmus"; ?>" />
```

When PHP is done with this text, it will read:

```
<input type="text" name="first_name" value="Rasmus" />
```

The PHP code within the opening and closing markers does not have to be on the same line. If the closing marker of a PHP instruction is the last thing on a line, the line break following the closing tag is removed as well. Thus, we can replace the PHP instructions in the "Hello, world" example with:

```
<?php
        echo "Hello, world"; ?>
<br />
```

with no change in the resulting HTML.

SGML Style :

SGML stands for Standard Generalized Markup Language The "classic" style of embedding PHP comes from SGML instruction processing tags. To use this method, simply enclose the PHP in <? and ?>. Here's the "Hello world" example again:

```
<? echo "Hello, world"; ?>
```

This style, known as short tags, is the shortest and least intrusive, and it can be turned off so as to not clash with the XML PI (Process Instruction) tag in the php.ini initialization file. Consequently, if you want to write fully portable PHP code that you are going to distribute to other people (who might have short tags turned off), you should use the longer <?php ... ?> tag style, which cannot be turned off. If you have no intention of distributing your code, you don't have an issue with telling people who want to use your code to turn on short tags, and you are not planning on mixing XML in with your PHP code, then using this tag style is okay.

ASP Style :

Because neither the SGML nor XML tag style is strictly legal HTML,[*] some HTML editors do not parse it correctly for color syntax highlighting, context-sensitive help, and other such niceties. Some will even go so far as to helpfully remove the "offending" code for you.

```
if( $a &gt; 5 )...?
```

However, many of these same HTML editors recognize another mechanism (no more legal than PHP's) for embedding code that of Microsoft's Active Server Pages (ASP). Like PHP, ASP is a method for embedding server-side scripts within documents.

If you want to use ASP-aware tools to edit files that contain embedded PHP, you can use ASP-style tags to identify PHP regions. The ASP-style tag is the same as the SGML-style tag, but with % instead of ?:

```
<% echo "Hello, world"; %>
```

In all other ways, the ASP-style tag works the same as the SGML-style tag.

ASP-style tags are not enabled by default. To use these tags, either build PHP with the --enable-asp-tags option or enable asp_tags in the PHP configuration file.

Script Style :

The final method of distinguishing PHP from HTML involves a tag invented to allow client-side scripting within HTML pages, the <script> tag. You might recognize it as the tag in which JavaScriptis embedded. Since PHP is processed and removed from the file before it reaches the browser, you can use the <script> tag to surround PHP code. To use this method, simply specify "php" as the value of the language attribute of the tag:

```
<script language="php">
        echo "Hello, world";
</script>
```

This method is most useful with HTML editors that work only on strictly legal HTML files and don't yet support XML processing commands.

Echoing Content Directly :

Perhaps the single most common operation within a PHP application is displaying data to the user. In the context of a web application, this means inserting into the HTML document information that will become HTML when viewed by the user.

To simplify this operation, PHP provides special versions of the SGML and ASP tags that automatically take the value inside the tag and insert it into the HTML page. To use this feature, add an equals sign(=) to the opening tag. With this technique, we can rewrite our form example as:

```
<input type="text" name="first_name" value="<?="Rasmus"; ?>">
```

If you have ASP-style tags enabled, you can do the same with your ASP tags:

```
<p>This number (<%= 2 + 2 %>)<br />
and this number (<% echo (2 + 2); %>) <br />
Are the same.</p>
```

After processing, the resulting HTML is:

```
<p>This number (4) <br />
and this number (4) <br />
are the same.</p>
```

Installation of Apache

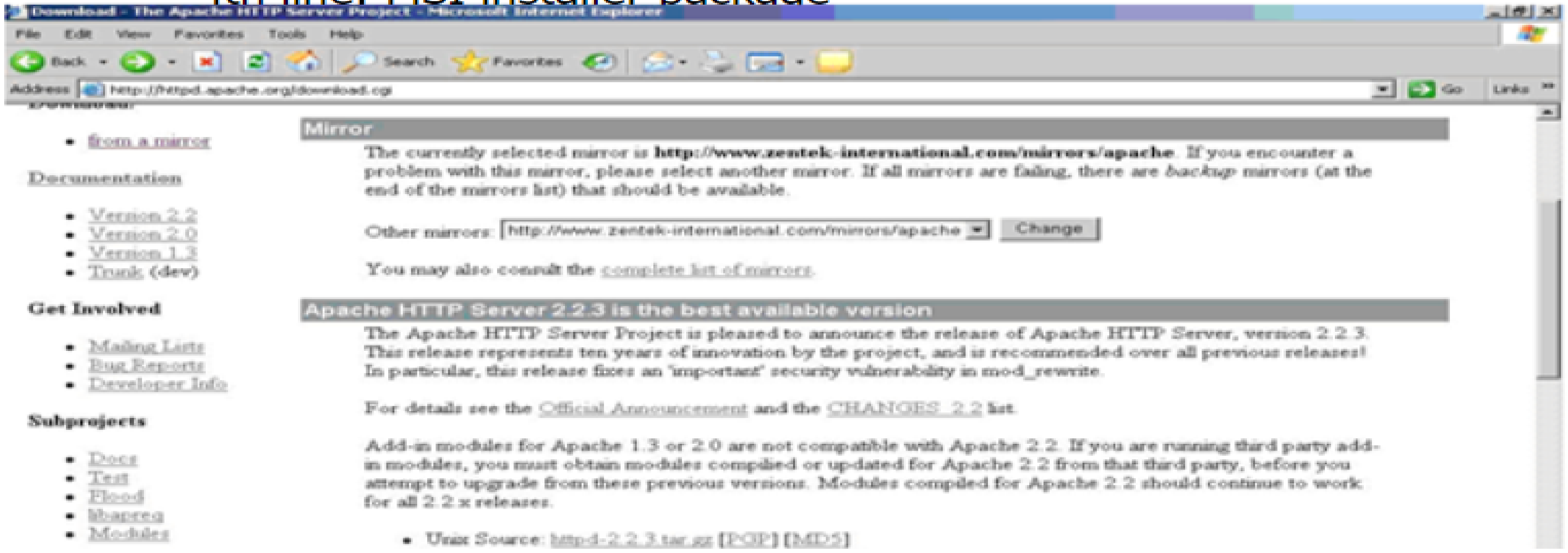
1. Download the Apache HTTP Server

The latest version is 2.2.3, download from Apache download page

(http://httpd.apache.org/download.cgi)

Windows: 3rd line: include source files, compile Apache source code

4th line: MSI installer package


Download - The Apache HTTP Server Project - Microsoft Internet Explorer

Address: http://httpd.apache.org/download.cgi

- from a mirror

Documentation

- Version 2.2
- Version 2.0
- Version 1.3
- Trunk (dev)

Get Involved

- Mailing Lists
- Bug Reports
- Developer Info

Subprojects

- Docs
- Test
- Flood
- libapreq
- Modules

Mirror

The currently selected mirror is **http://www.zentek-international.com/mirrors/apache**. If you encounter a problem with this mirror, please select another mirror. If all mirrors are failing, there are *backup* mirrors (at the end of the mirrors list) that should be available.

Other mirrors: http://www.zentek-international.com/mirrors/apache Change

You may also consult the complete list of mirrors.

Apache HTTP Server 2.2.3 is the best available version

The Apache HTTP Server Project is pleased to announce the release of Apache HTTP Server, version 2.2.3. This release represents ten years of innovation by the project, and is recommended over all previous releases! In particular, this release fixes an 'important' security vulnerability in mod_rewrite.

For details see the Official Announcement and the CHANGES_2.2 list.

Add-in modules for Apache 1.3 or 2.0 are not compatible with Apache 2.2. If you are running third party add-in modules, you must obtain modules compiled or updated for Apache 2.2 from that third party, before you attempt to upgrade from these previous versions. Modules compiled for Apache 2.2 should continue to work for all 2.2.x releases.

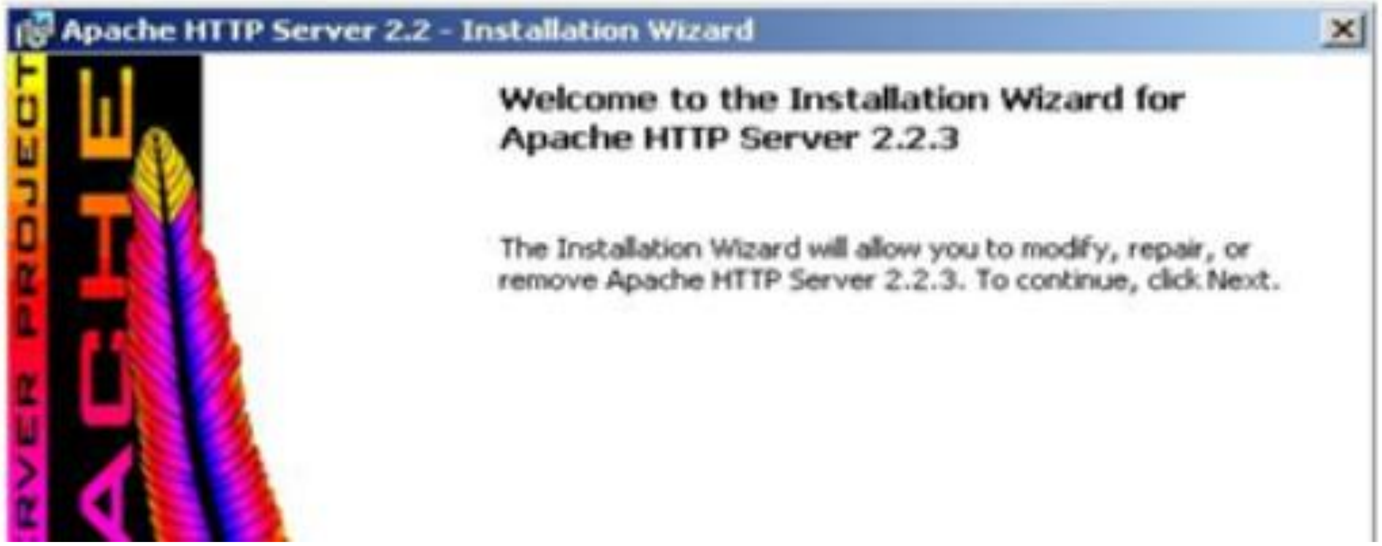
- Unix Source: httpd-2.2.3.tar.gz [PGP] [MD5]

Installation of Apache

2. After you download the installer, double-click on the file

apache_2.2.3-win32-x86-no_ssl.msi

start the installation process. You will get a welcome screen, as shown next:

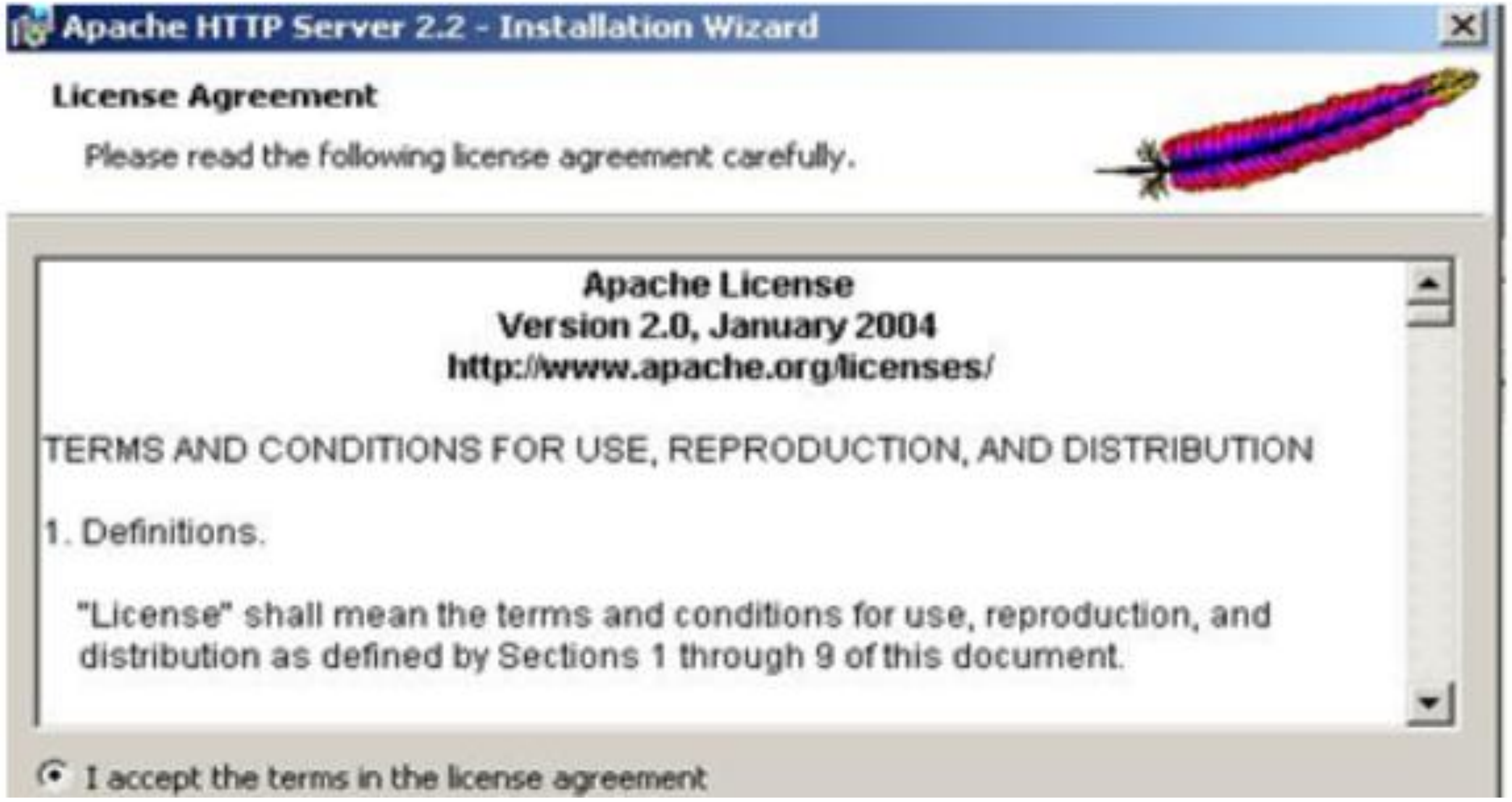


Welcome to the Installation Wizard for Apache HTTP Server 2.2.3

The Installation Wizard will allow you to modify, repair, or remove Apache HTTP Server 2.2.3. To continue, click Next.

Installation of Apache

- 3. the Apache license screen



License Agreement

Please read the following license agreement carefully.

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

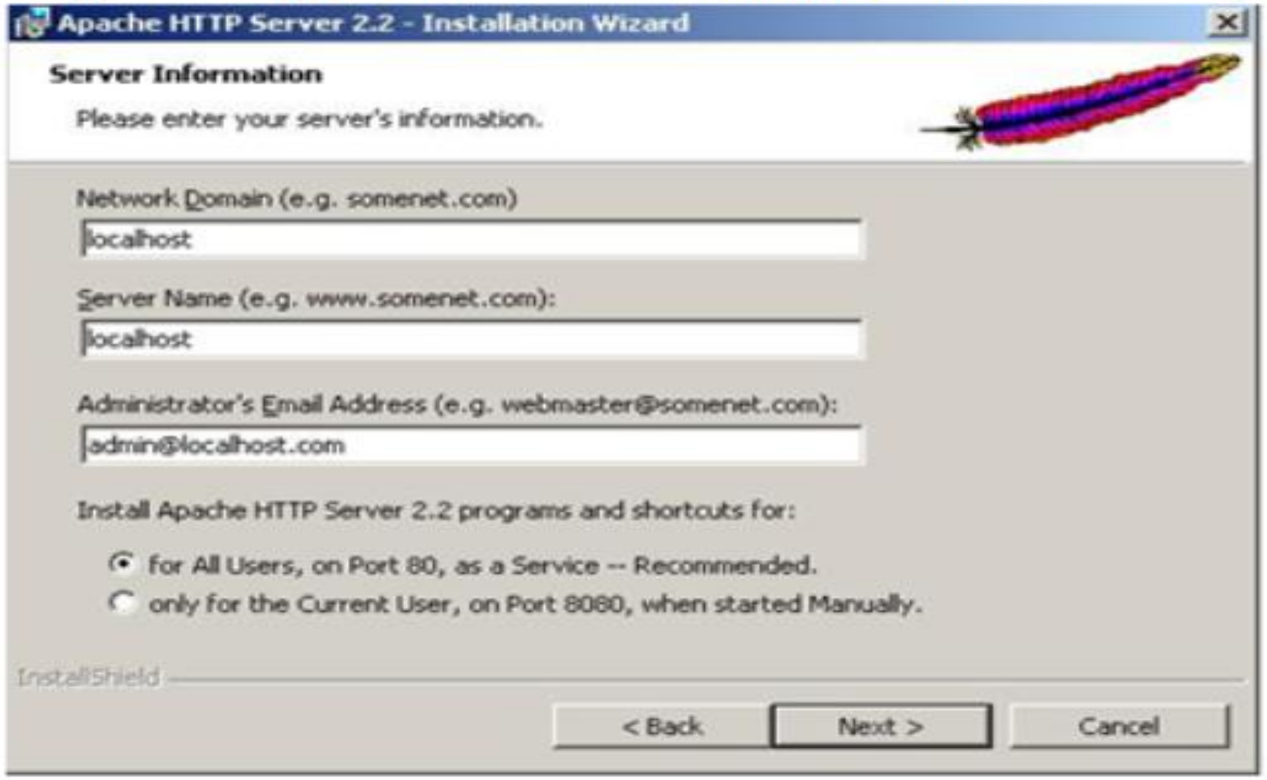
1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

[✓ I accept the terms in the license agreement]

Installation of Apache

- 4. asks you to provide basic information about your computer



Apache HTTP Server 2.2 - Installation Wizard

Server Information

Please enter your server's information.

Network Domain (e.g. somenet.com)
localhost

Server Name (e.g. www.somenet.com):
localhost

Administrator's Email Address (e.g. webmaster@somenet.com):
admin@localhost.com

Install Apache HTTP Server 2.2 programs and shortcuts for:

- [✓ for All Users, on Port 80, as a Service -- Recommended.]
- only for the Current User, on Port 8080, when started Manually.

If your machine does not have a full network address, use localhost or 127.0.0.1 as the server name

Configuration of Apache

- Apache keeps all of its configuration information in files.

- Directories:
  - ***Conf*** main configuration file ***httpd.conf***
  - ***Htdocs*** hold your web server

- Httpd.conf

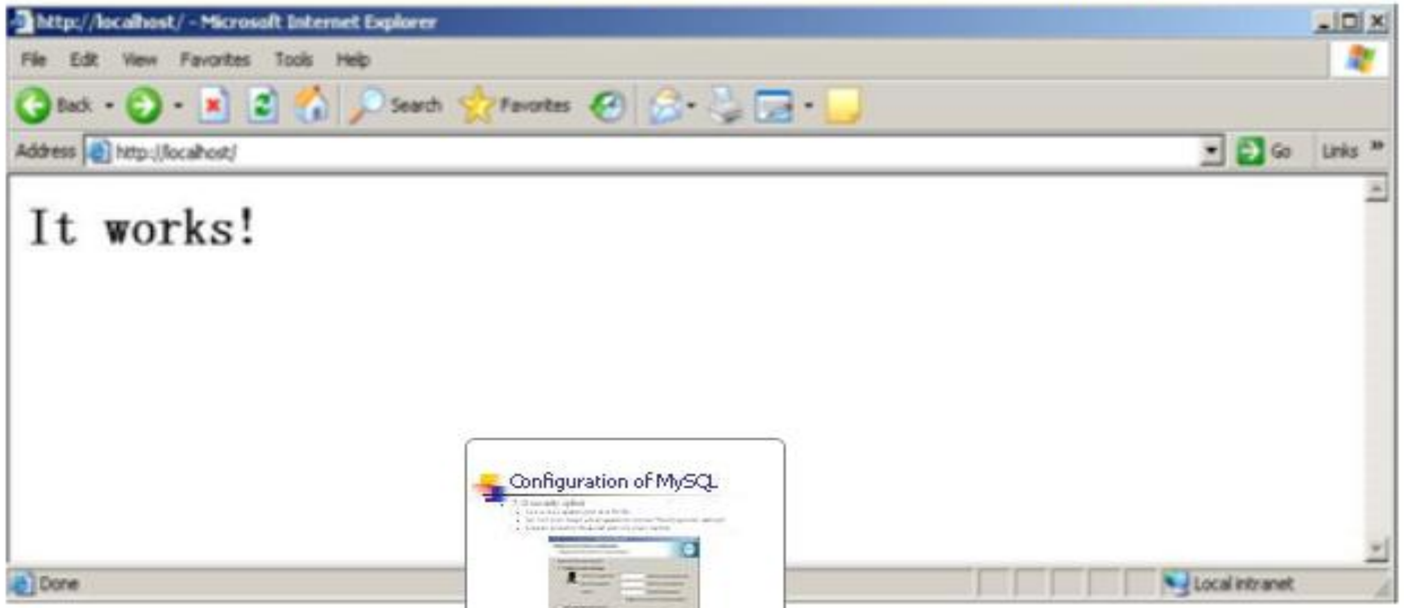
1. Directives : configure specific settings of Apache, such as authorization, performance, and network parameters
2. Containers: specify the context to which those settings refer.

Test of Apache Server

- Open the apache server

  programs->Apache HTTP Server 2.2.3->Control Apache Server->Start
- If everything goes well, you can access Apache using a browser.

  http://localhost/
- The default installation page will be displayed

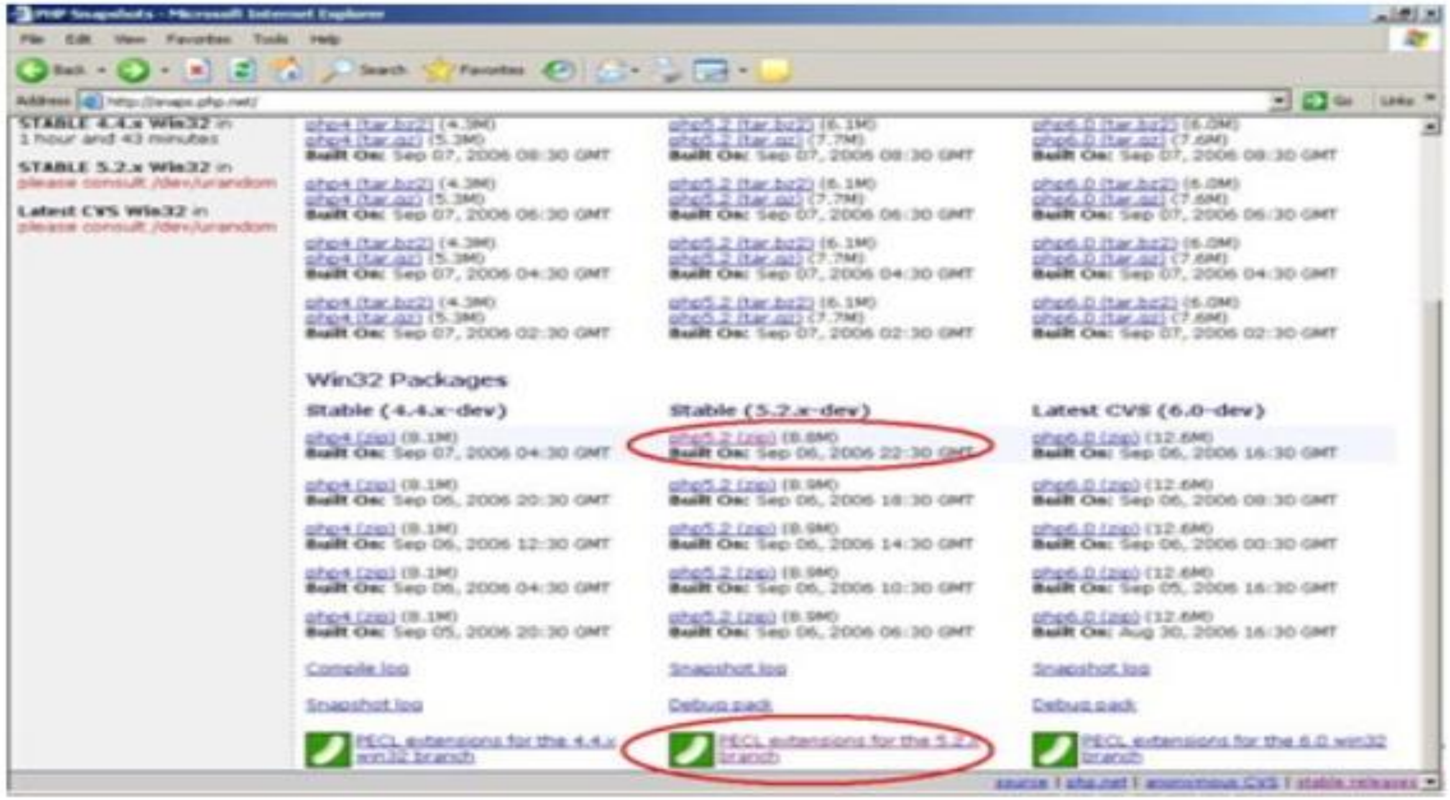


It works!

Configuration of MySQL

Installation of PHP

- 1. Download PHP package(http://snaps.php.net)



PHP Snapshots - Microsoft Internet Explorer

Address http://snaps.php.net/

STABLE 4.4.x Win32 in
1 hour and 43 minutes

STABLE 5.2.x Win32 in
please consult /dev/urandom

Latest CVS Win32 in
please consult /dev/urandom

php4 (tar.bz2) (4.3M) php4 (tar.gz) (5.3M) **Built On:** Sep 07, 2006 08:30 GMT	php5.2 (tar.bz2) (6.1M) php5.2 (tar.gz) (7.7M) **Built On:** Sep 07, 2006 08:30 GMT	php6.0 (tar.bz2) (6.0M) php6.0 (tar.gz) (7.6M) **Built On:** Sep 07, 2006 08:30 GMT
php4 (tar.bz2) (4.3M) php4 (tar.gz) (5.3M) **Built On:** Sep 07, 2006 06:30 GMT	php5.2 (tar.bz2) (6.1M) php5.2 (tar.gz) (7.7M) **Built On:** Sep 07, 2006 06:30 GMT	php6.0 (tar.bz2) (6.0M) php6.0 (tar.gz) (7.6M) **Built On:** Sep 07, 2006 06:30 GMT
php4 (tar.bz2) (4.3M) php4 (tar.gz) (5.3M) **Built On:** Sep 07, 2006 04:30 GMT	php5.2 (tar.bz2) (6.1M) php5.2 (tar.gz) (7.7M) **Built On:** Sep 07, 2006 04:30 GMT	php6.0 (tar.bz2) (6.0M) php6.0 (tar.gz) (7.6M) **Built On:** Sep 07, 2006 04:30 GMT
php4 (tar.bz2) (4.3M) php4 (tar.gz) (5.3M) **Built On:** Sep 07, 2006 02:30 GMT	php5.2 (tar.bz2) (6.1M) php5.2 (tar.gz) (7.7M) **Built On:** Sep 07, 2006 02:30 GMT	php6.0 (tar.bz2) (6.0M) php6.0 (tar.gz) (7.6M) **Built On:** Sep 07, 2006 02:30 GMT

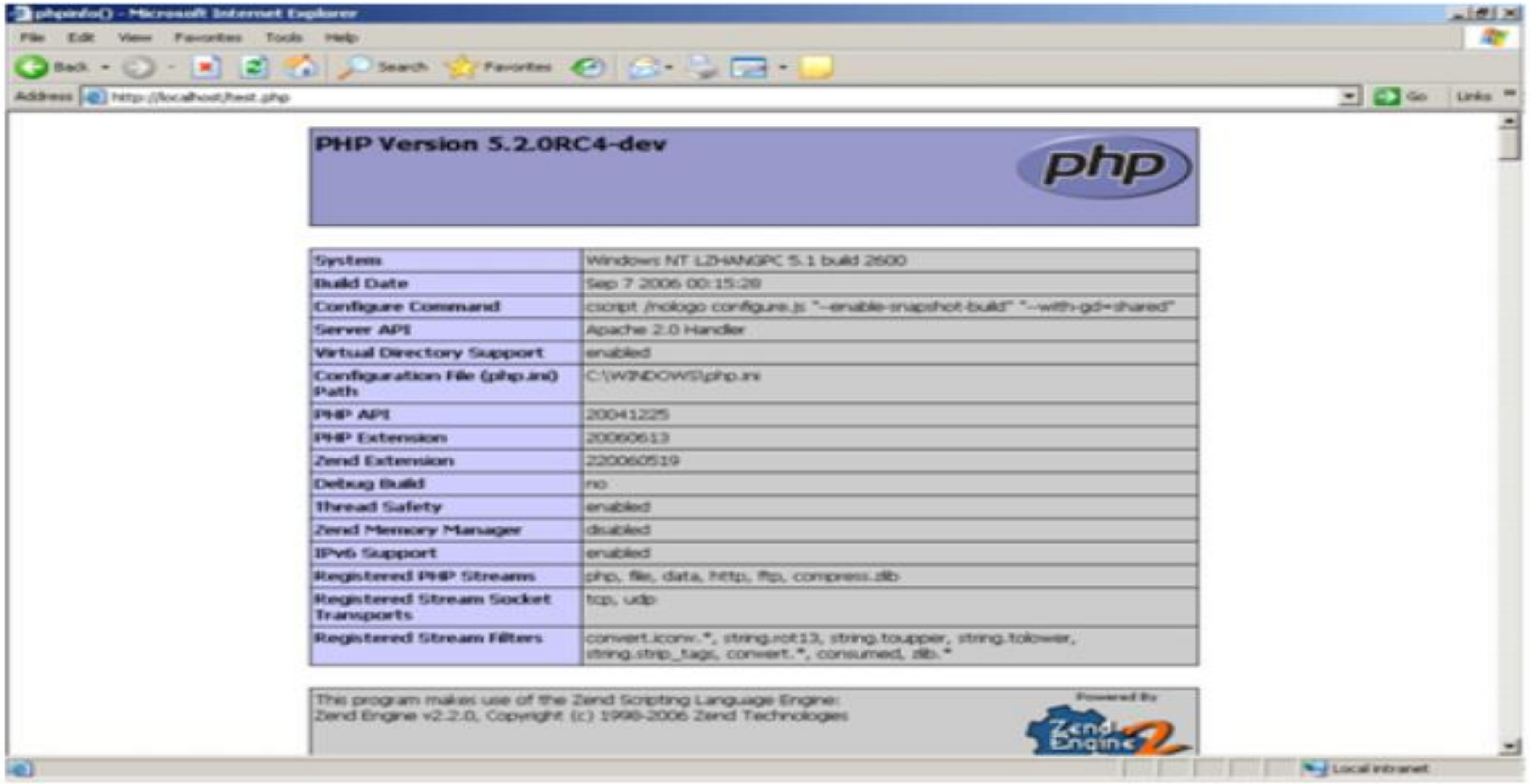
Win32 Packages

Stable (4.4.x-dev)	Stable (5.2.x-dev)	Latest CVS (6.0-dev)
php4 (zip) (9.1M) **Built On:** Sep 07, 2006 04:30 GMT	php5.2 (zip) (8.8M) **Built On:** Sep 06, 2006 22:30 GMT	php6.0 (zip) (12.6M) **Built On:** Sep 06, 2006 16:30 GMT
php4 (zip) (9.1M) **Built On:** Sep 06, 2006 20:30 GMT	php5.2 (zip) (8.9M) **Built On:** Sep 06, 2006 18:30 GMT	php6.0 (zip) (12.6M) **Built On:** Sep 06, 2006 08:30 GMT
php4 (zip) (9.1M) **Built On:** Sep 06, 2006 12:30 GMT	php5.2 (zip) (8.9M) **Built On:** Sep 06, 2006 14:30 GMT	php6.0 (zip) (12.6M) **Built On:** Sep 06, 2006 00:30 GMT
php4 (zip) (9.1M) **Built On:** Sep 06, 2006 04:30 GMT	php5.2 (zip) (8.9M) **Built On:** Sep 06, 2006 10:30 GMT	php6.0 (zip) (12.6M) **Built On:** Sep 05, 2006 16:30 GMT
php4 (zip) (9.1M) **Built On:** Sep 05, 2006 20:30 GMT	php5.2 (zip) (8.9M) **Built On:** Sep 06, 2006 06:30 GMT	php6.0 (zip) (12.6M) **Built On:** Aug 30, 2006 16:30 GMT
Compile log	Snapshot log	Snapshot log
Snapshot log	Debug pack	Debug pack
PECL extensions for the 4.4.x win32 branch	PECL extensions for the 5.2.x branch	PECL extensions for the 6.0 win32 branch

source \| php.net \| anonymous CVS \| stable releases

Be compatible with apache 2.2.x, download version of (5.2.x)
PECL extension: support more extensions

Test of Apache & PHP

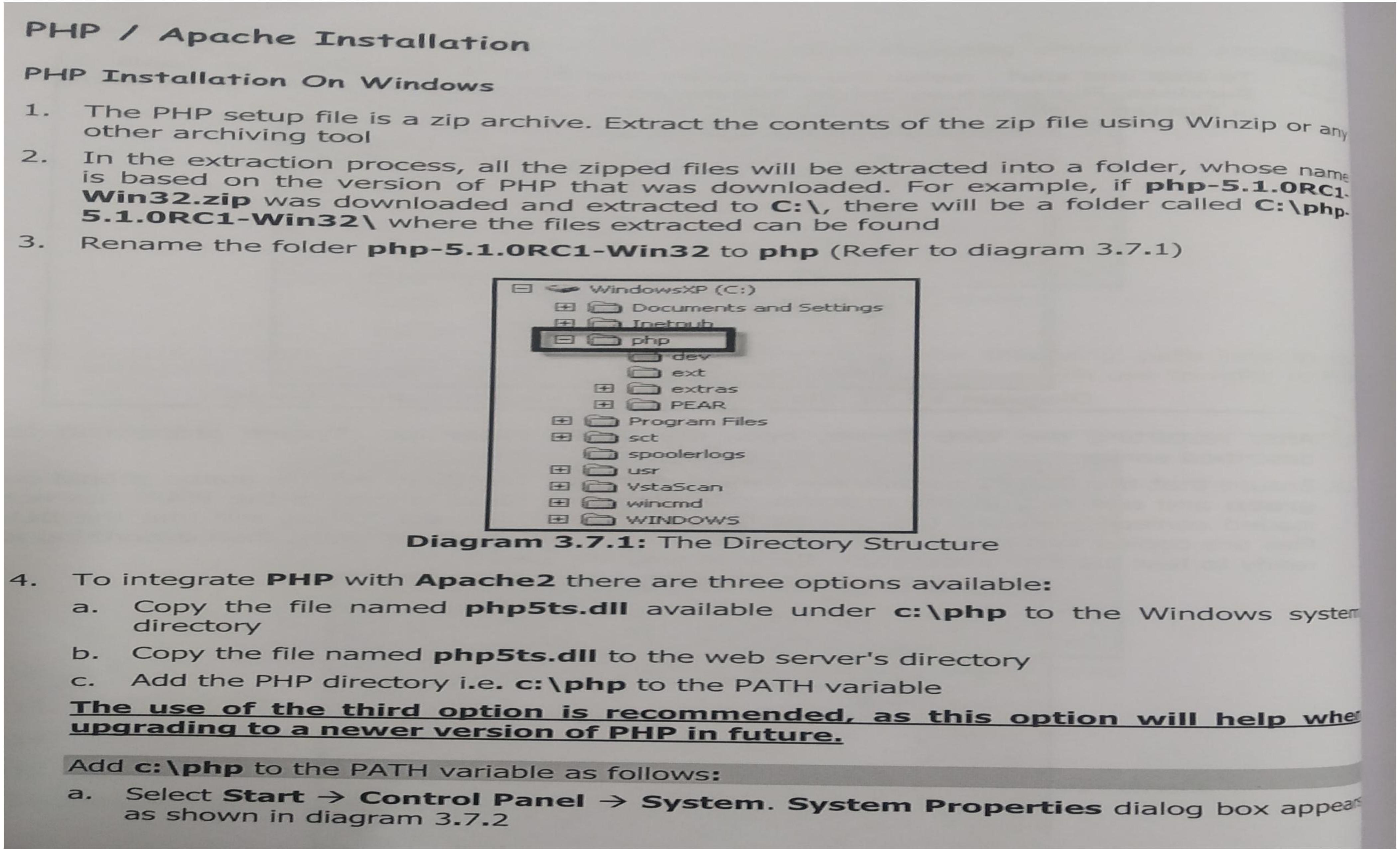

phpinfo() - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/test.php
Go Links
Local intranet

PHP Version 5.2.0RC4-dev

System	Windows NT LZHANGPC 5.1 build 2600
Build Date	Sep 7 2006 00:15:28
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--with-gd=shared"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS\php.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	enabled
Zend Memory Manager	disabled
IPv6 Support	enabled
Registered PHP Streams	php, file, data, http, ftp, compress.zlib
Registered Stream Socket Transports	tcp, udp
Registered Stream Filters	convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, zlib.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v2.2.0, Copyright (c) 1998-2006 Zend Technologies

Powered By



PHP / Apache Installation

PHP Installation On Windows

1. The PHP setup file is a zip archive. Extract the contents of the zip file using Winzip or any other archiving tool
2. In the extraction process, all the zipped files will be extracted into a folder, whose name is based on the version of PHP that was downloaded. For example, if **php-5.1.0RC1-Win32.zip** was downloaded and extracted to **C:\**, there will be a folder called **C:\php-5.1.0RC1-Win32\** where the files extracted can be found
3. Rename the folder **php-5.1.0RC1-Win32** to **php** (Refer to diagram 3.7.1)

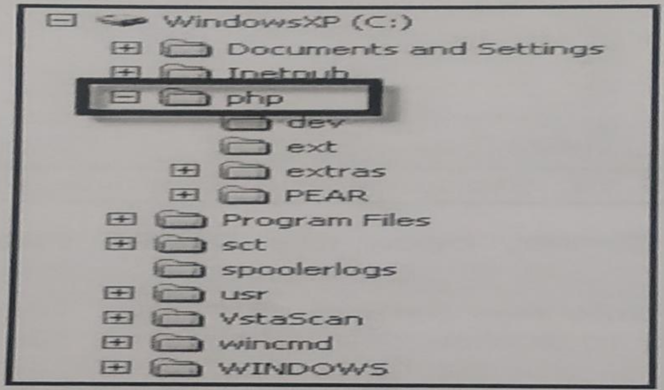


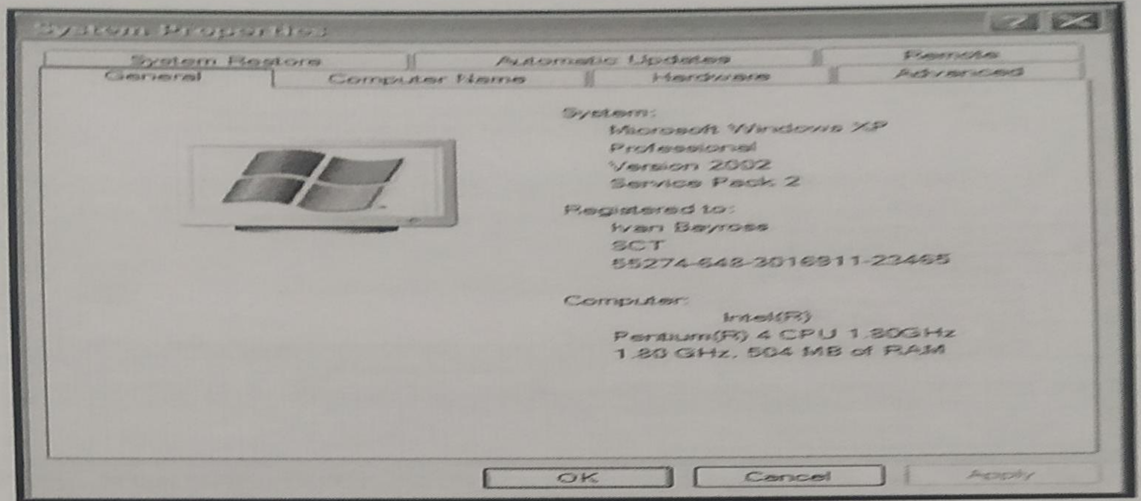
Diagram 3.7.1: The Directory Structure

4. To integrate **PHP** with **Apache2** there are three options available:
   a. Copy the file named **php5ts.dll** available under **c:\php** to the Windows syster directory
   b. Copy the file named **php5ts.dll** to the web server's directory
   c. Add the PHP directory i.e. **c:\php** to the PATH variable

   <u>The use of the third option is recommended, as this option will help wher upgrading to a newer version of PHP in future.</u>

Add **c:\php** to the PATH variable as follows:

a. Select **Start → Control Panel → System**. **System Properties** dialog box appear as shown in diagram 3.7.2



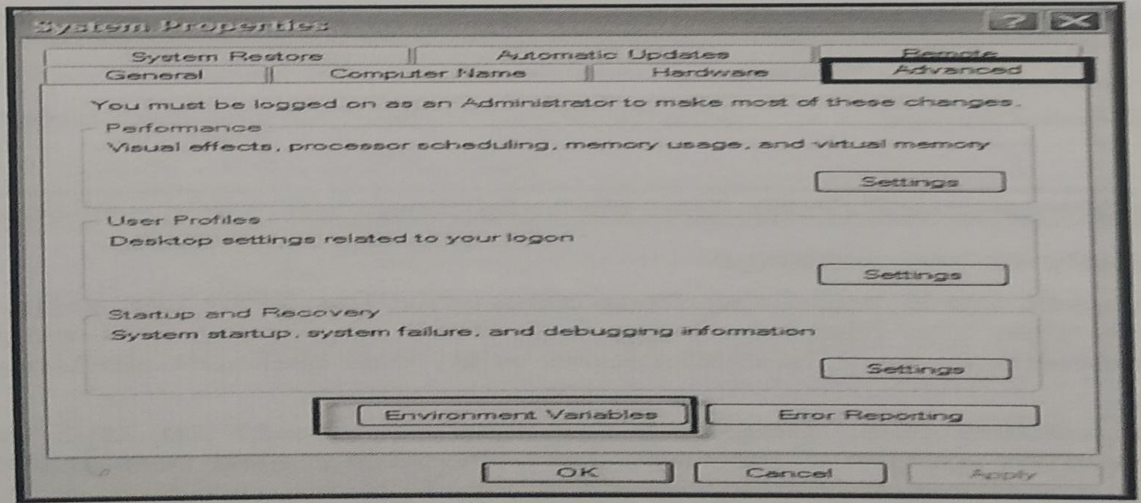
System:
Microsoft Windows XP
Professional
Version 2002
Service Pack 2

Registered to:
Ivan Bayross
SCT
55274-648-3016811-23465

Computer:
Intel(R)
Pentium(R) 4 CPU 1.80GHz
1.80 GHz, 504 MB of RAM

Diagram 3.7.2: The System Properties in control panel

b. Select the **Advanced** tab. Refer to diagram 3.7.3



You must be logged on as an Administrator to make most of these changes.

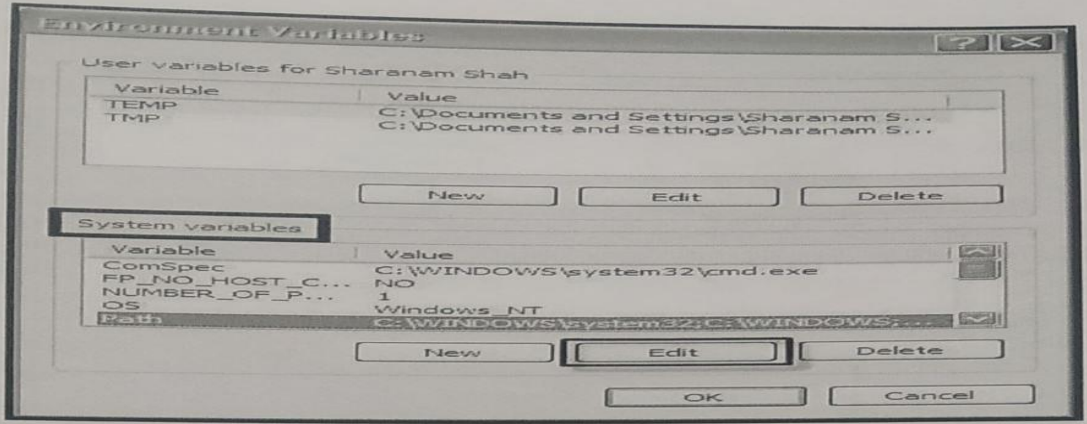
Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your logon

Startup and Recovery
System startup, system failure, and debugging information

Diagram 3.7.3: The System Properties Advanced Tab

c. Click the **Environment Variables** button. This pops up a window as shown in diagram 3.7.4



User variables for Sharanam Shah

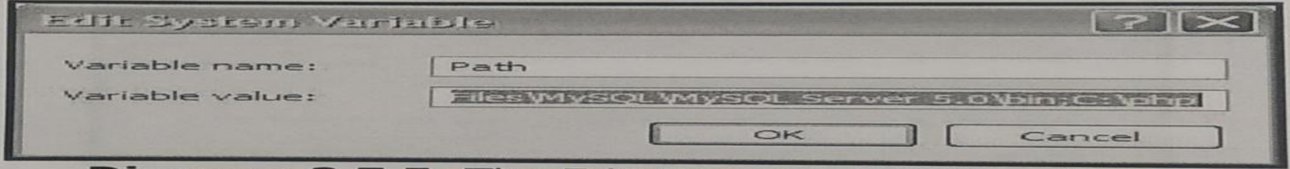
Variable	Value
TEMP	C:\Documents and Settings\Sharanam S...
TMP	C:\Documents and Settings\Sharanam S...

System variables

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	1
OS	Windows_NT
Path	C:\WINDOWS\system32;C:\WINDOWS;...

Diagram 3.7.4: The System Properties Environment Variables

d. Select the **Path** variable under **System variables** section and click **Edit**. This pops up a window, in the text box **Variable value** add **;C:\php** at the end of the path as shown in diagram 3.7.5



Variable name: Path

Variable value: Files\MySQL\MySQL Server 5.0\bin;C:\php

Diagram 3.7.5: The Edit System variable window

e. Click **OK** to apply these settings

5. The next step is to set up a valid configuration file for PHP i.e. **php.ini**. There are two in **files** distributed in the zip file i.e. **php.ini-dist** and **php.ini-recommended**. Use th **php.ini-recommended**, as it is optimized with the default settings for performance an security

6. Copy the chosen ini-file (i.e. **php.ini-recommended**) to the windows directory (i.e c:\windows **or** c:\winnt as the case may be) directory and rename it to **php.ini**

REMINDER

If the file system in use is NTFS on Windows NT, 2000, XP or 2003, make sure that the user running the web server has read permissions to the **php.ini**.

PHP is now setup on the Windows system.

3. SETTING UP AND CONFIGURING PHP TO WORK UNDER APACHE AND IIS

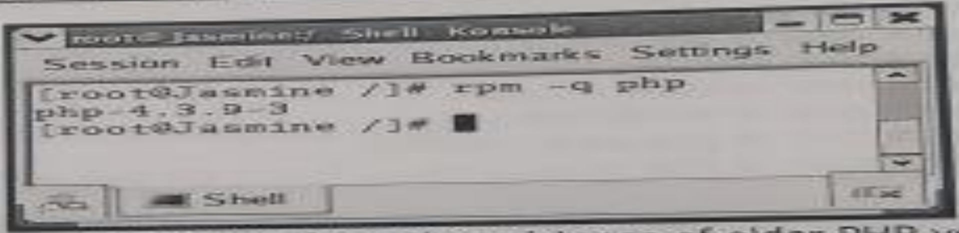
PHP Installation On Linux

Login as root.

Erasing An Older Version

All flavor of Linux come bundled with some stable version of PHP. If a full install is the choice when setting up the Linux box, then some version of PHP definitely exists on the Linux flavor in use. Hence prior installing the latest version of PHP, it is prudent to verify the existence of any older version of PHP on the Linux box. This can be done as follows: (Refer to diagram 3.8.1)

```
<System Prompt> rpm -q php
```


root@Jasmine:/ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

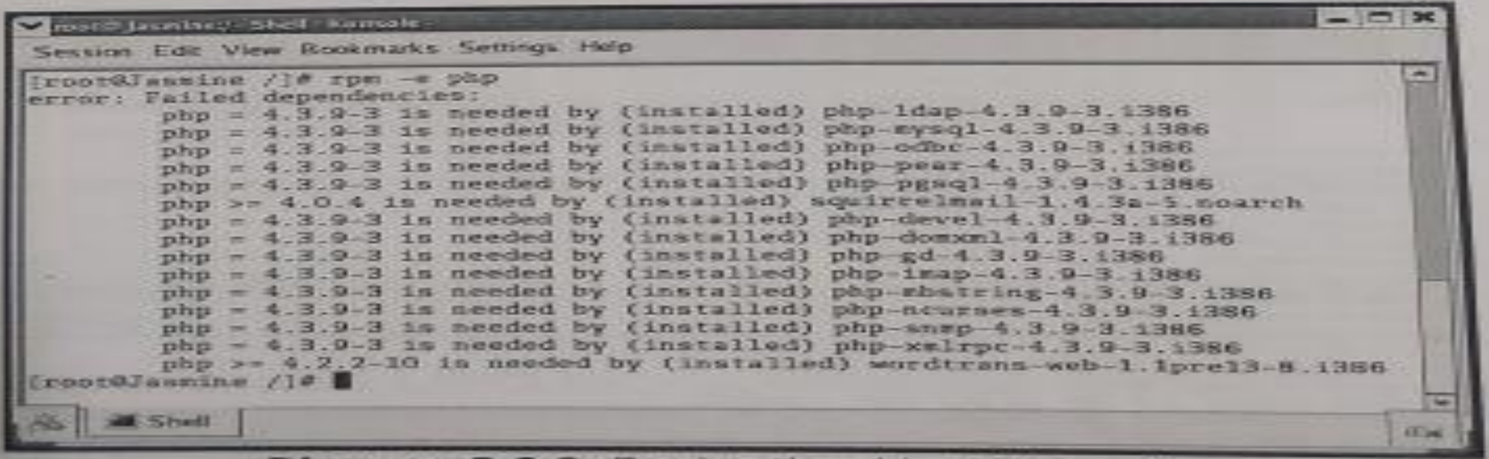
```
[root@Jasmine /]# rpm -q php
php-4.3.9-3
[root@Jasmine /]#
```

Diagram 3.8.1: Verifying the existence of older PHP version

The above diagram indicates that PHP version 4.3.9-3 exists.

Since this is an older version, erase it as follows: (Refer to diagram 3.8.2)

```
<System Prompt> rpm -e php
```


root@Jasmine:/ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

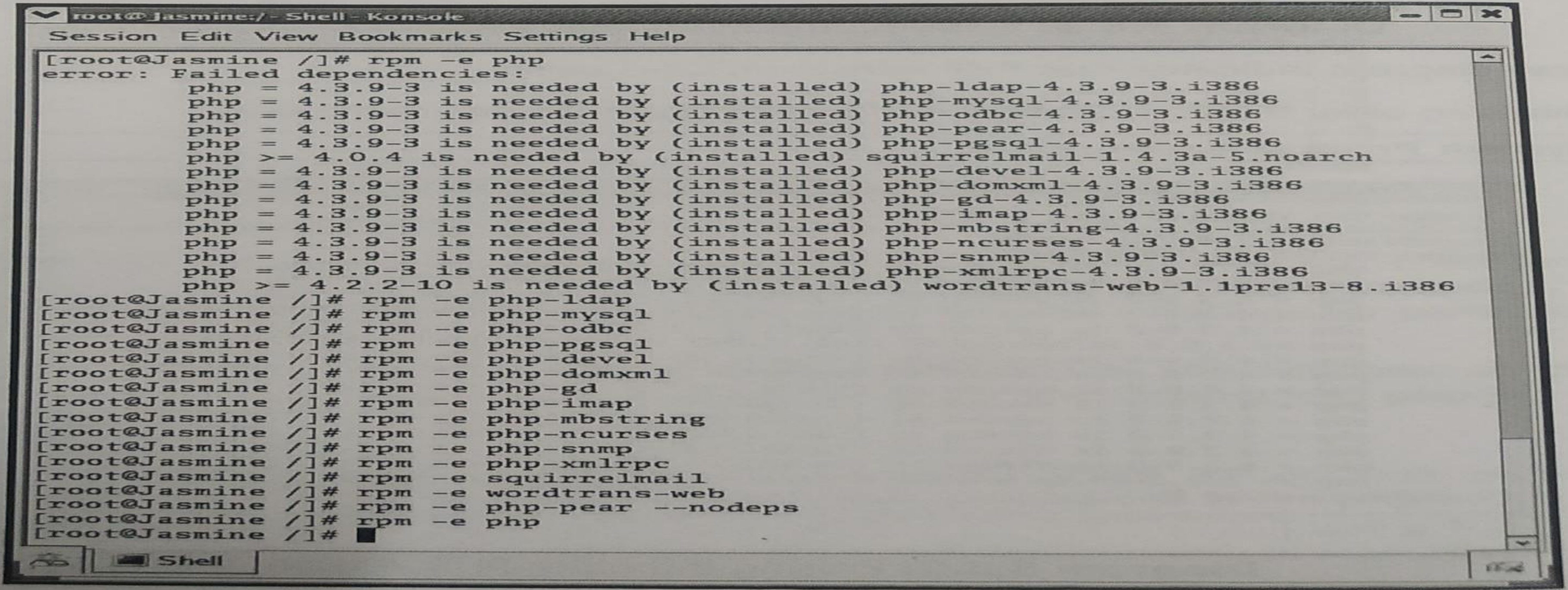
```
[root@Jasmine /]# rpm -e php
error: Failed dependencies:
        php = 4.3.9-3 is needed by (installed) php-ldap-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-mysql-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-odbc-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-pear-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-pgsql-4.3.9-3.i386
        php >= 4.0.4 is needed by (installed) squirrelmail-1.4.3a-5.noarch
        php = 4.3.9-3 is needed by (installed) php-devel-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-domxml-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-gd-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-imap-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-mbstring-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-ncurses-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-snmp-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-xmlrpc-4.3.9-3.i386
        php >= 4.2.2-10 is needed by (installed) wordtrans-web-1.1pre13-8.i386
[root@Jasmine /]#
```

Diagram 3.8.2: Erasing the older PHP version

The above diagram shows a list of dependencies, which exist as a support system to th current PHP version. These **need to be** uninstalled (erased) prior uninstalling (erasing) PHP.

Erase these dependencies as: (Refer to diagram 3.8.3)

```
<System Prompt> rpm -e php-ldap
<System Prompt> rpm -e php-mysql
<System Prompt> rpm -e php-odbc
<System Prompt> rpm -e php-pgsql
<System Prompt> rpm -e php-devel
<System Prompt> rpm -e php-domxml
<System Prompt> rpm -e php-gd
<System Prompt> rpm -e php-imap
<System Prompt> rpm -e php-mbstring
<System Prompt> rpm -e php-ncurses
<System Prompt> rpm -e php-snmp
<System Prompt> rpm -e php-xmlrpc
<System Prompt> rpm -e squirrelmail
<System Prompt> rpm -e wordtrans-web
<System Prompt> rpm -e php-pear --nodeps
<System Prompt> rpm -e php
```


root@Jasmine:/ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
[root@Jasmine /]# rpm -e php
error: Failed dependencies:
        php = 4.3.9-3 is needed by (installed) php-ldap-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-mysql-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-odbc-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-pear-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-pgsql-4.3.9-3.i386
        php >= 4.0.4 is needed by (installed) squirrelmail-1.4.3a-5.noarch
        php = 4.3.9-3 is needed by (installed) php-devel-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-domxml-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-gd-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-imap-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-mbstring-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-ncurses-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-snmp-4.3.9-3.i386
        php = 4.3.9-3 is needed by (installed) php-xmlrpc-4.3.9-3.i386
        php >= 4.2.2-10 is needed by (installed) wordtrans-web-1.1pre13-8.i386
[root@Jasmine /]# rpm -e php-ldap
[root@Jasmine /]# rpm -e php-mysql
[root@Jasmine /]# rpm -e php-odbc
[root@Jasmine /]# rpm -e php-pgsql
[root@Jasmine /]# rpm -e php-devel
[root@Jasmine /]# rpm -e php-domxml
[root@Jasmine /]# rpm -e php-gd
[root@Jasmine /]# rpm -e php-imap
[root@Jasmine /]# rpm -e php-mbstring
[root@Jasmine /]# rpm -e php-ncurses
[root@Jasmine /]# rpm -e php-snmp
[root@Jasmine /]# rpm -e php-xmlrpc
[root@Jasmine /]# rpm -e squirrelmail
[root@Jasmine /]# rpm -e wordtrans-web
[root@Jasmine /]# rpm -e php-pear --nodeps
[root@Jasmine /]# rpm -e php
[root@Jasmine /]# █
```

Diagram 3.8.3: Erasing the older PHP version

3. SETTING UP AND CONFIGURING PHP TO WORK UNDER APACHE AND IIS

If any of the listed dependencies **are required** then skip the erasing of that particular dependency and erase PHP as follows:

```
<System Prompt> rpm -e php --nodeps
i.e. two hyphens followed by nodeps
```

In the above command, **--nodeps** instructs the rpm package manager to ignore the existence of the dependencies and erase the current version of PHP.

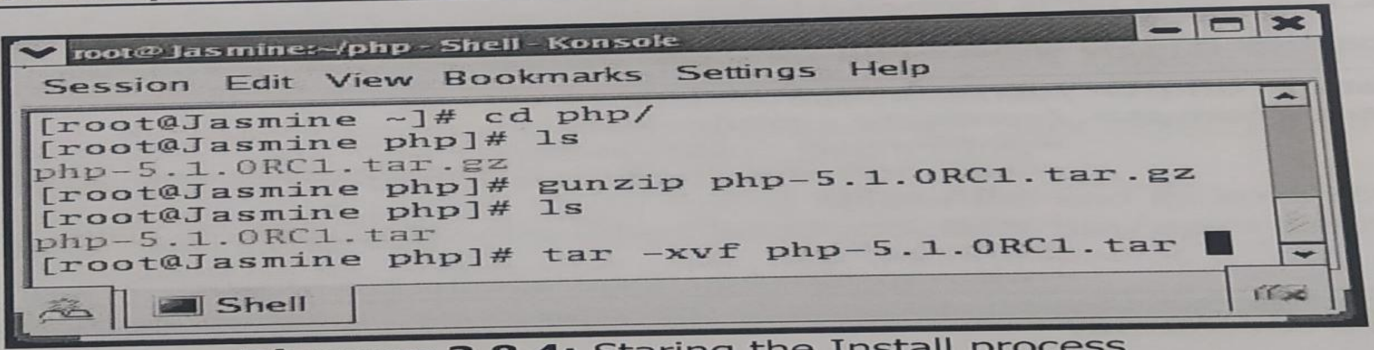
Installing PHP From Its Source File

It is always advisable to install PHP using its source (tarball). This provides access to quicker releases updates instead of waiting for some entity to release an rpm for a particular flavor of Linux.

1. Make a directory **/php**. Copy the downloaded file **php-5.1.0RC1.tar.gz** here

2. Unzip the file as: (Refer to diagram 3.8.4)

```
<System Prompt> gunzip php-5.1.0RC1.tar.gz
```


root@Jasmine:~/php - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
[root@Jasmine ~]# cd php/
[root@Jasmine php]# ls
php-5.1.0RC1.tar.gz
[root@Jasmine php]# gunzip php-5.1.0RC1.tar.gz
[root@Jasmine php]# ls
php-5.1.0RC1.tar
[root@Jasmine php]# tar -xvf php-5.1.0RC1.tar
```

Diagram 3.8.4: Staring the Install process

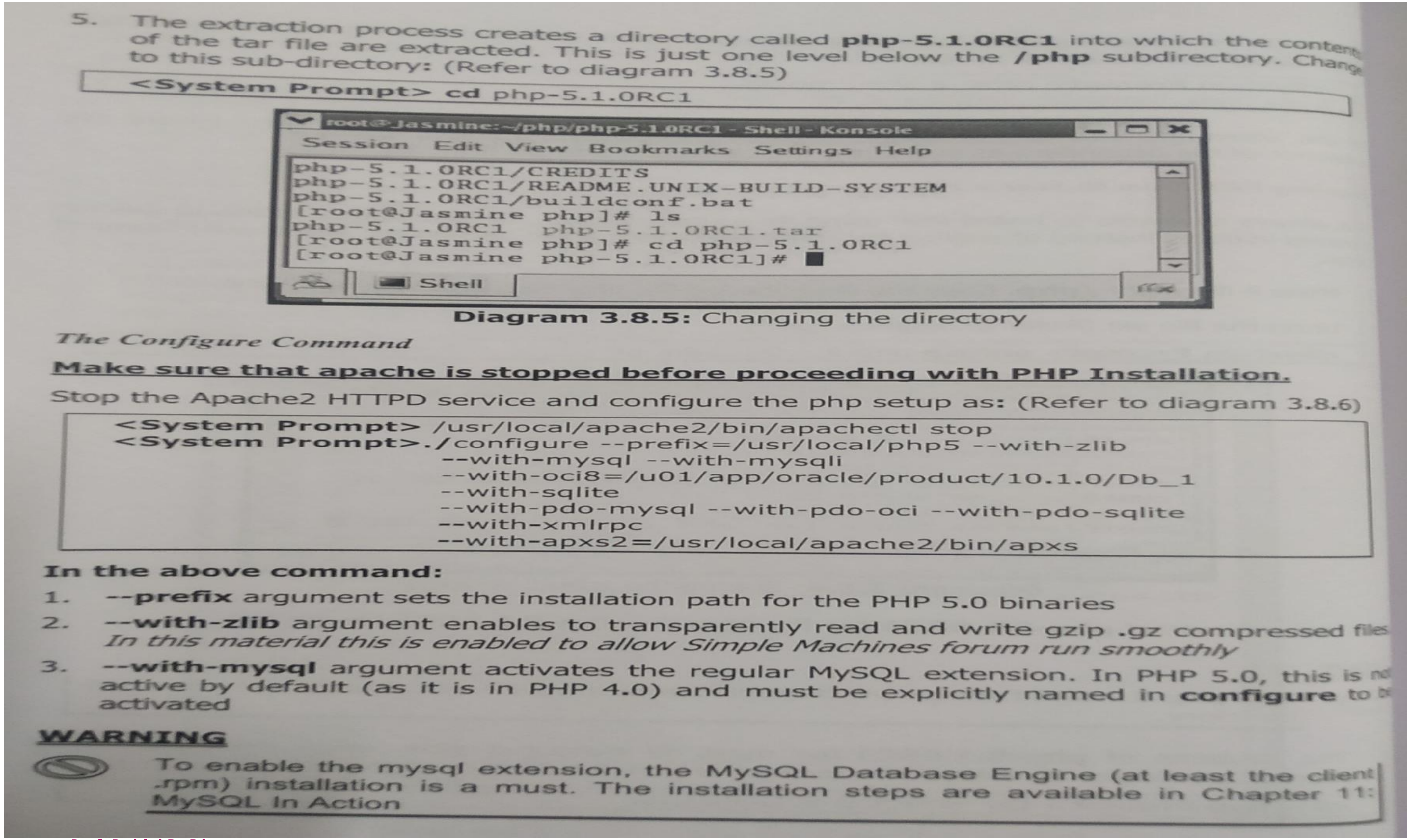
3. This will extract **php-5.1.0RC1.tar** file from the **php-5.1.0RC1.tar.gz** file

HINT

The **php-5.1.0RC1.tar.gz** will be replaced by **php-5.1.0RC1.tar** in the same sub directory.

4. The contents of **php-5.1.0RC1.tar** must be extracted after which the actual install process can begin. To extract the content of php-5.1.0RC1.tar: (Refer to diagram 3.8.4)

```
<System Prompt> tar -xvf php-5.1.0RC1.tar
```



5. The extraction process creates a directory called **php-5.1.0RC1** into which the conten of the tar file are extracted. This is just one level below the **/php** subdirectory. Chang to this sub-directory: (Refer to diagram 3.8.5)

```
<System Prompt> cd php-5.1.0RC1
```

root@Jasmine:~/php/php-5.1.0RC1 - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
php-5.1.0RC1/CREDITS
php-5.1.0RC1/README.UNIX-BUILD-SYSTEM
php-5.1.0RC1/buildconf.bat
[root@Jasmine php]# ls
php-5.1.0RC1  php-5.1.0RC1.tar
[root@Jasmine php]# cd php-5.1.0RC1
[root@Jasmine php-5.1.0RC1]# █
```

Diagram 3.8.5: Changing the directory

The Configure Command

Make sure that apache is stopped before proceeding with PHP Installation.

Stop the Apache2 HTTPD service and configure the php setup as: (Refer to diagram 3.8.6)

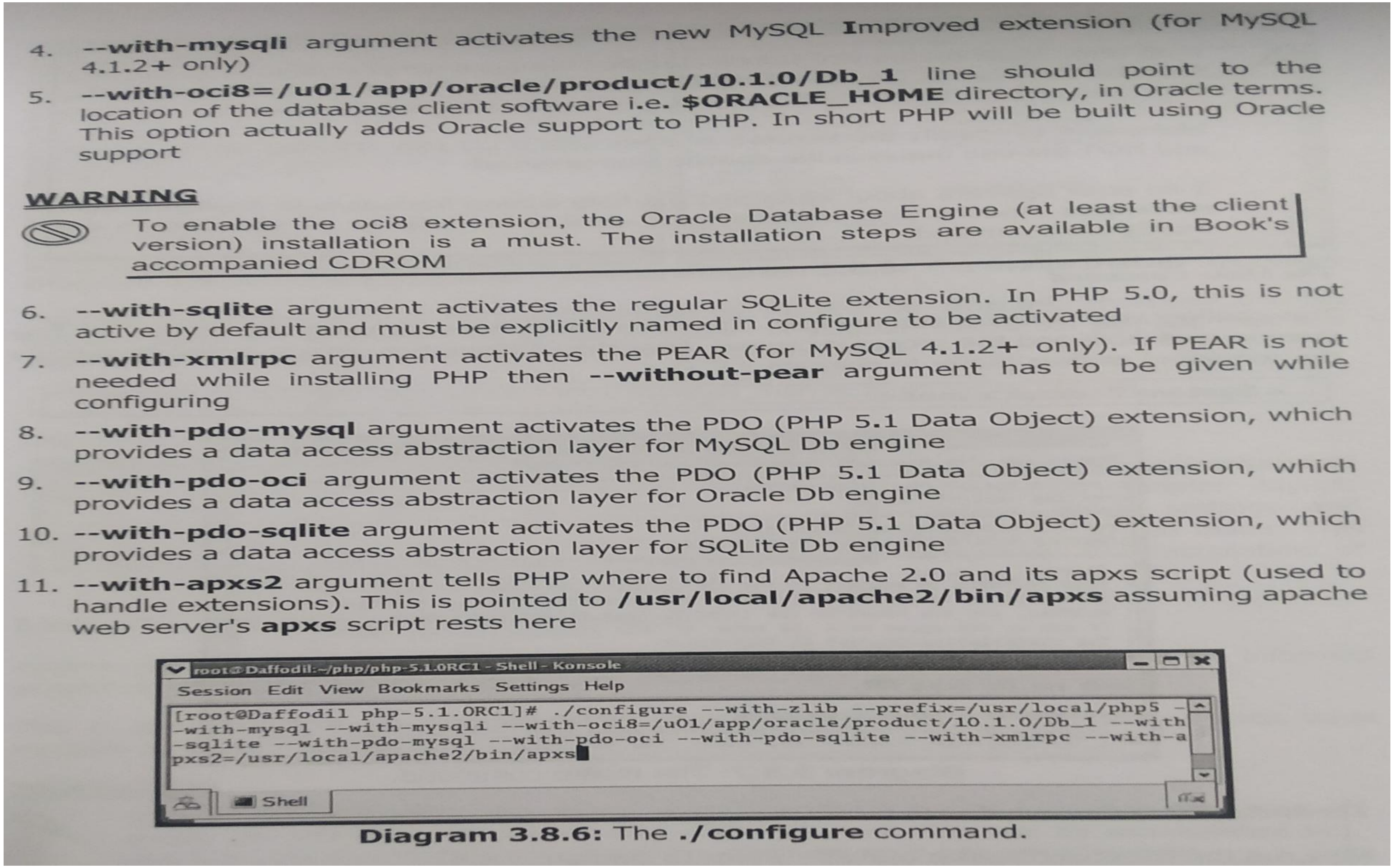
```
<System Prompt> /usr/local/apache2/bin/apachectl stop
<System Prompt>./configure --prefix=/usr/local/php5 --with-zlib
                --with-mysql --with-mysqli
                --with-oci8=/u01/app/oracle/product/10.1.0/Db_1
                --with-sqlite
                --with-pdo-mysql --with-pdo-oci --with-pdo-sqlite
                --with-xmlrpc
                --with-apxs2=/usr/local/apache2/bin/apxs
```

In the above command:

1. **--prefix** argument sets the installation path for the PHP 5.0 binaries
2. **--with-zlib** argument enables to transparently read and write gzip .gz compressed files
   In this material this is enabled to allow Simple Machines forum run smoothly
3. **--with-mysql** argument activates the regular MySQL extension. In PHP 5.0, this is no active by default (as it is in PHP 4.0) and must be explicitly named in **configure** to b activated

WARNING

To enable the mysql extension, the MySQL Database Engine (at least the client .rpm) installation is a must. The installation steps are available in Chapter 11: MySQL In Action

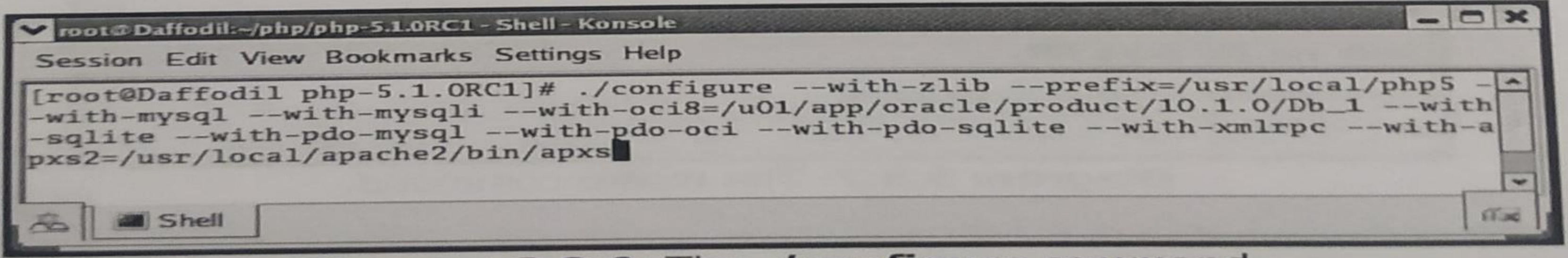


4. **--with-mysqli** argument activates the new MySQL **Improved** extension (for MySQL 4.1.2+ only)
5. **--with-oci8=/u01/app/oracle/product/10.1.0/Db_1** line should point to the location of the database client software i.e. **$ORACLE_HOME** directory, in Oracle terms. This option actually adds Oracle support to PHP. In short PHP will be built using Oracle support

WARNING

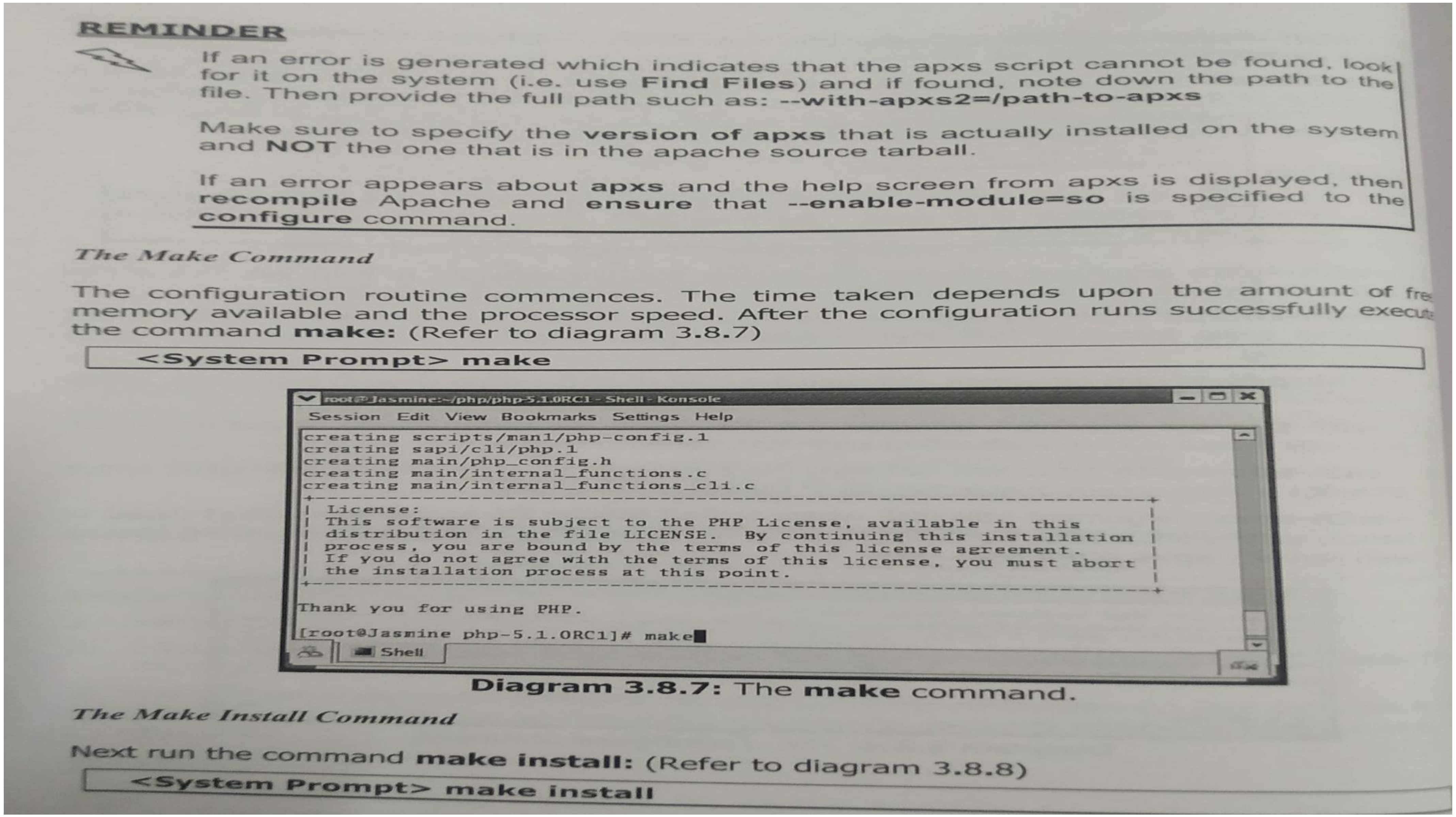
To enable the oci8 extension, the Oracle Database Engine (at least the client version) installation is a must. The installation steps are available in Book's accompanied CDROM

6. **--with-sqlite** argument activates the regular SQLite extension. In PHP 5.0, this is not active by default and must be explicitly named in configure to be activated
7. **--with-xmlrpc** argument activates the PEAR (for MySQL 4.1.2+ only). If PEAR is not needed while installing PHP then **--without-pear** argument has to be given while configuring
8. **--with-pdo-mysql** argument activates the PDO (PHP 5.1 Data Object) extension, which provides a data access abstraction layer for MySQL Db engine
9. **--with-pdo-oci** argument activates the PDO (PHP 5.1 Data Object) extension, which provides a data access abstraction layer for Oracle Db engine
10. **--with-pdo-sqlite** argument activates the PDO (PHP 5.1 Data Object) extension, which provides a data access abstraction layer for SQLite Db engine
11. **--with-apxs2** argument tells PHP where to find Apache 2.0 and its apxs script (used to handle extensions). This is pointed to **/usr/local/apache2/bin/apxs** assuming apache web server's **apxs** script rests here


root@Daffodil:~/php/php-5.1.0RC1 - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
[root@Daffodil php-5.1.0RC1]# ./configure --with-zlib --prefix=/usr/local/php5 -
-with-mysql --with-mysqli --with-oci8=/u01/app/oracle/product/10.1.0/Db_1 --with
-sqlite --with-pdo-mysql --with-pdo-oci --with-pdo-sqlite --with-xmlrpc --with-a
pxs2=/usr/local/apache2/bin/apxs
```

Diagram 3.8.6: The **./configure** command.



REMINDER

If an error is generated which indicates that the apxs script cannot be found, look for it on the system (i.e. use **Find Files**) and if found, note down the path to the file. Then provide the full path such as: **--with-apxs2=/path-to-apxs**

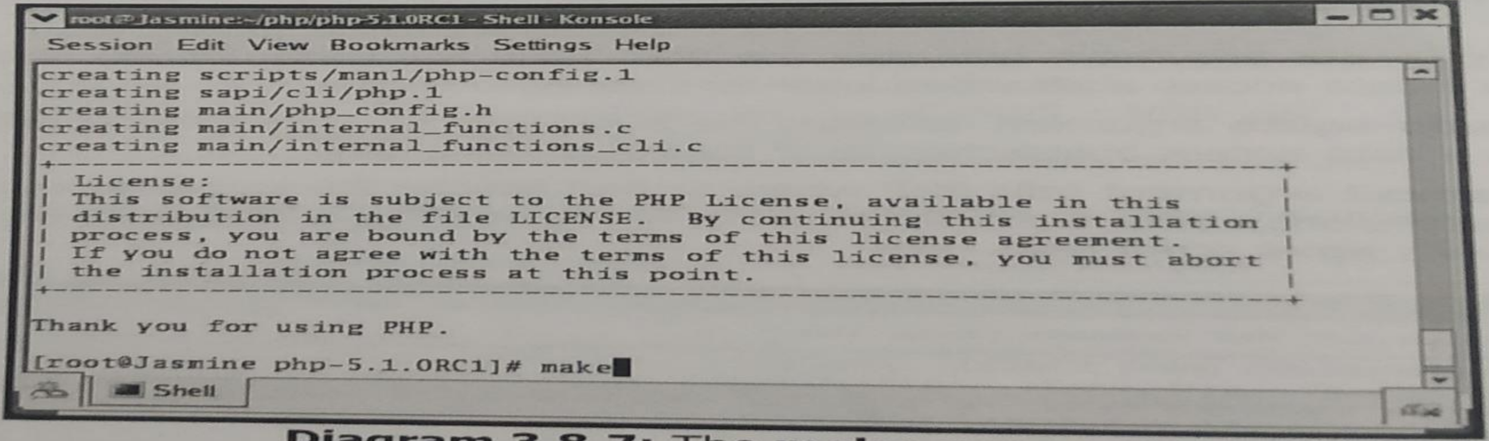
Make sure to specify the **version of apxs** that is actually installed on the system and **NOT** the one that is in the apache source tarball.

If an error appears about **apxs** and the help screen from apxs is displayed, then **recompile** Apache and **ensure** that **--enable-module=so** is specified to the **configure** command.

The Make Command

The configuration routine commences. The time taken depends upon the amount of fre memory available and the processor speed. After the configuration runs successfully execut the command **make:** (Refer to diagram 3.8.7)

```
<System Prompt> make
```


root@Jasmine:~/php/php-5.1.0RC1 - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

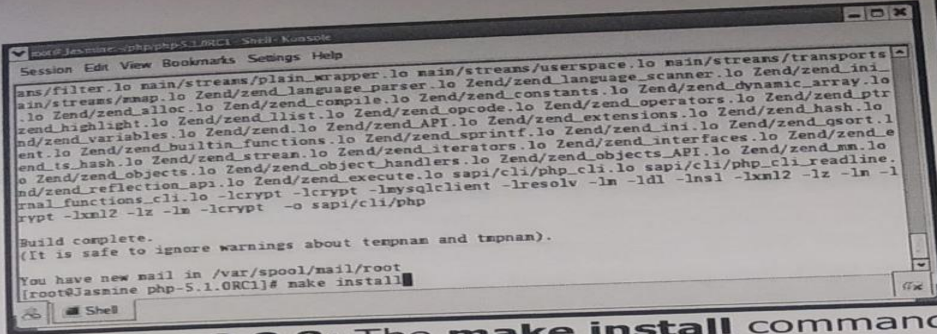
```
creating scripts/man1/php-config.1
creating sapi/cli/php.1
creating main/php_config.h
creating main/internal_functions.c
creating main/internal_functions_cli.c
+--------------------------------------------------------------------+
| License:                                                           |
| This software is subject to the PHP License, available in this    |
| distribution in the file LICENSE.  By continuing this installation |
| process, you are bound by the terms of this license agreement.     |
| If you do not agree with the terms of this license, you must abort |
| the installation process at this point.                            |
+--------------------------------------------------------------------+

Thank you for using PHP.

[root@Jasmine php-5.1.0RC1]# make
```

Diagram 3.8.7: The **make** command.

The Make Install Command

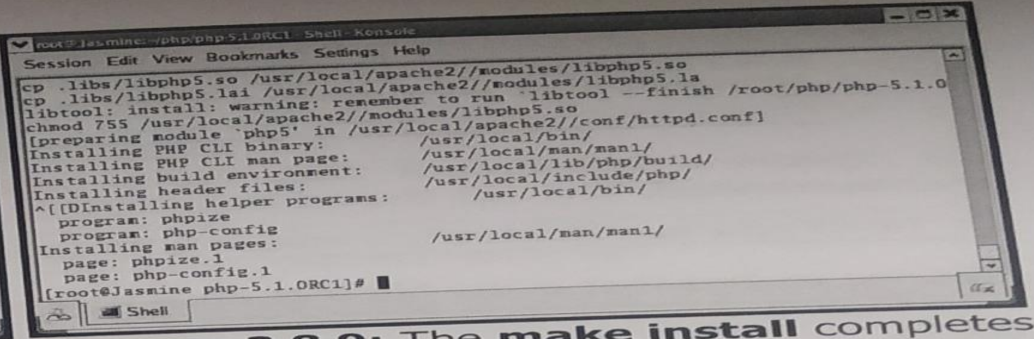
Next run the command **make install:** (Refer to diagram 3.8.8)

```
<System Prompt> make install
```


root@Jasmine:~/php/php-5.1.0RC1 - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
ans/filter.lo main/streams/plain_wrapper.lo main/streams/userspace.lo main/streams/transports
ain/streams/mmap.lo Zend/zend_language_parser.lo Zend/zend_language_scanner.lo Zend/zend_ini_
.lo Zend/zend_alloc.lo Zend/zend_compile.lo Zend/zend_constants.lo Zend/zend_dynamic_array.lo
zend_highlight.lo Zend/zend_llist.lo Zend/zend_opcode.lo Zend/zend_operators.lo Zend/zend_ptr
nd/zend_variables.lo Zend/zend.lo Zend/zend_API.lo Zend/zend_extensions.lo Zend/zend_hash.lo
ent.lo Zend/zend_builtin_functions.lo Zend/zend_sprintf.lo Zend/zend_ini.lo Zend/zend_qsort.l
end_ts_hash.lo Zend/zend_stream.lo Zend/zend_iterators.lo Zend/zend_interfaces.lo Zend/zend_e
o Zend/zend_objects.lo Zend/zend_object_handlers.lo Zend/zend_objects_API.lo Zend/zend_mm.lo
nd/zend_reflection_api.lo Zend/zend_execute.lo sapi/cli/php_cli.lo sapi/cli/php_cli_readline.
rnal_functions_cli.lo -lcrypt -lcrypt -lmysqlclient -lresolv -lm -ldl -lnsl -lxml2 -lz -lm -l
rypt -lxml2 -lz -lm -lcrypt  -o sapi/cli/php

Build complete.
(It is safe to ignore warnings about tempnam and tmpnam).

You have new mail in /var/spool/mail/root
[root@Jasmine php-5.1.0RC1]# make install
```

Diagram 3.8.8: The **make install** command


root@Jasmine:~/php/php-5.1.0RC1 - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

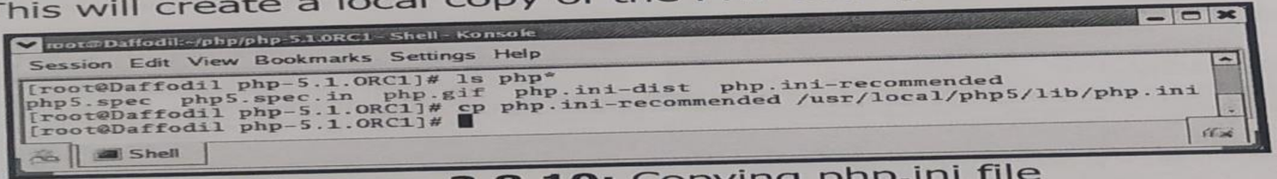
```
cp .libs/libphp5.so /usr/local/apache2//modules/libphp5.so
cp .libs/libphp5.lai /usr/local/apache2//modules/libphp5.la
libtool: install: warning: remember to run `libtool --finish /root/php/php-5.1.0
chmod 755 /usr/local/apache2//modules/libphp5.so
[preparing module `php5' in /usr/local/apache2//conf/httpd.conf]
Installing PHP CLI binary:        /usr/local/bin/
Installing PHP CLI man page:      /usr/local/man/man1/
Installing build environment:     /usr/local/lib/php/build/
Installing header files:          /usr/local/include/php/
Installing helper programs:           /usr/local/bin/
^[[DInstalling helper programs:
  program: phpize
  program: php-config
Installing man pages:             /usr/local/man/man1/
  page: phpize.1
  page: php-config.1
[root@Jasmine php-5.1.0RC1]#
```

Diagram 3.8.9: The **make install** completes

The next step is to set up a valid configuration file for PHP i.e. **php.ini**. There are two **ini** files distributed in the source file (.tar.gz) as shown in diagram 3.8.10 i.e. **php.ini-dist** and **php.ini-recommended**. Use the file **php.ini-recommended**.

```
<System Prompt> cp php.ini-recommended /usr/local/php5/lib/php.ini
```

This will create a local copy of the PHP configuration file.


root@Daffodil:~/php/php-5.1.0RC1 - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
[root@Daffodil php-5.1.0RC1]# ls php*
php5.spec  php5.spec.in  php.gif  php.ini-dist  php.ini-recommended
[root@Daffodil php-5.1.0RC1]# cp php.ini-recommended /usr/local/php5/lib/php.ini
[root@Daffodil php-5.1.0RC1]#
```

Diagram 3.8.10: Copying php.ini file

The **php.ini-recommended** file has a simpler layout, contains fewer settings and allows PHP to run faster. This completes the installation of PHP on Linux.

Binding The PHP Installation With Apache2

Apache does not know that PHP is just installed. Therefore Apache needs to be informed about PHP especially where to find it.

This is done via Apache's httpd.conf file. Apache reads this file and understands what modules need to be loaded and where these modules are located.

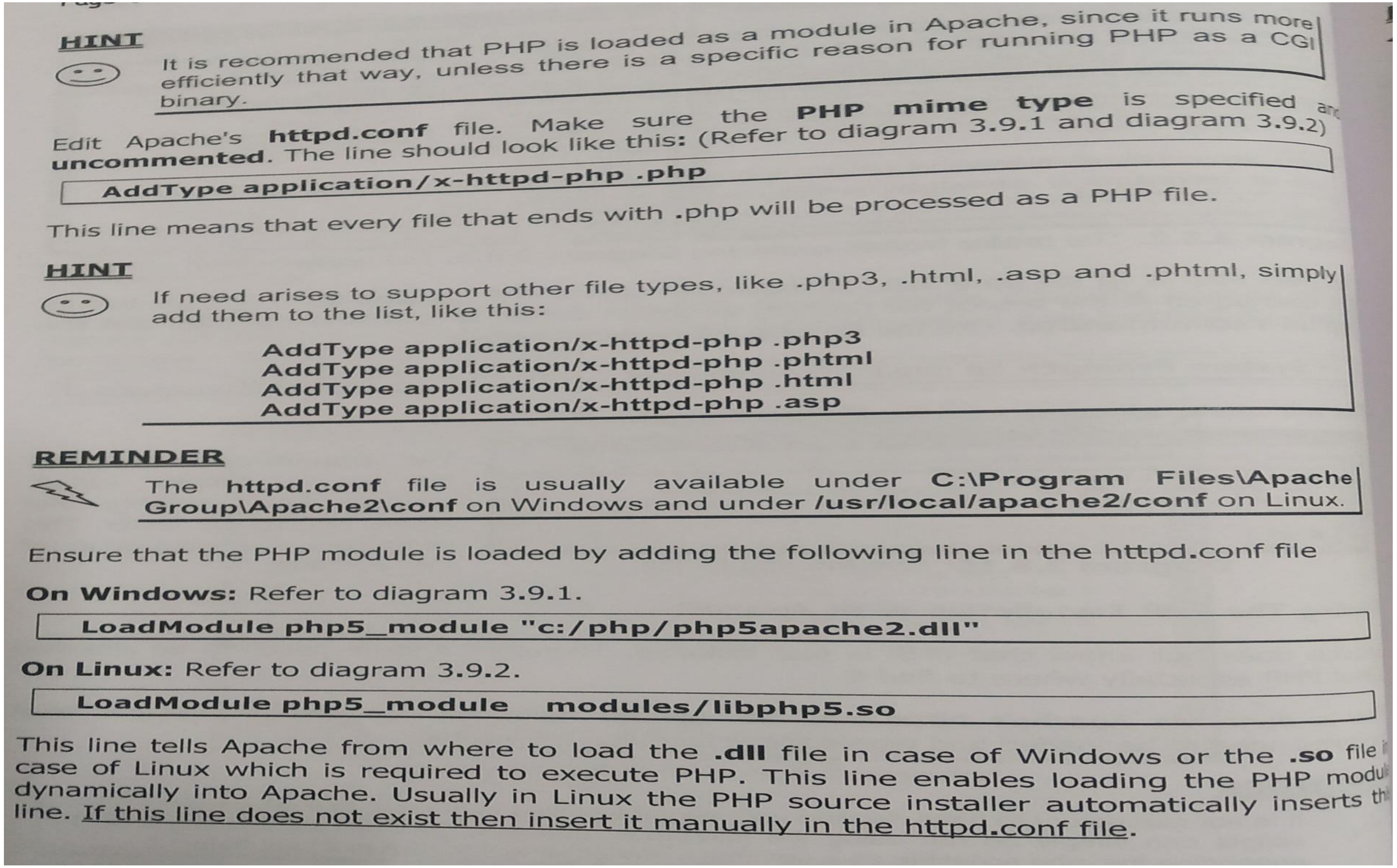
REMINDER

It is not mandatory to have Apache web server installed in order to test .php scripts. These scripts can simply be run using the interpreter **php.exe**. This can be accomplished by appending the .php script file as a command line argument to **php.exe** interpreter.

There are **two ways** to configure Apache to use PHP 5.

One is to configure it to load the PHP interpreter as an Apache module.

The other is to configure it to run the PHP interpreter as a CGI binary.



HINT

It is recommended that PHP is loaded as a module in Apache, since it runs more efficiently that way, unless there is a specific reason for running PHP as a CGI binary.

Edit Apache's **httpd.conf** file. Make sure the **PHP mime type** is specified an **uncommented**. The line should look like this: (Refer to diagram 3.9.1 and diagram 3.9.2)

```
AddType application/x-httpd-php .php
```

This line means that every file that ends with .php will be processed as a PHP file.

HINT

If need arises to support other file types, like .php3, .html, .asp and .phtml, simply add them to the list, like this:

```
AddType application/x-httpd-php .php3
AddType application/x-httpd-php .phtml
AddType application/x-httpd-php .html
AddType application/x-httpd-php .asp
```

REMINDER

The **httpd.conf** file is usually available under **C:\Program Files\Apache Group\Apache2\conf** on Windows and under **/usr/local/apache2/conf** on Linux.

Ensure that the PHP module is loaded by adding the following line in the httpd.conf file

On Windows: Refer to diagram 3.9.1.

```
LoadModule php5_module "c:/php/php5apache2.dll"
```

On Linux: Refer to diagram 3.9.2.

```
LoadModule php5_module   modules/libphp5.so
```

This line tells Apache from where to load the **.dll** file in case of Windows or the **.so** file i case of Linux which is required to execute PHP. This line enables loading the PHP modul dynamically into Apache. Usually in Linux the PHP source installer automatically inserts th line. <u>If this line does not exist then insert it manually in the httpd.conf file</u>.

REMINDER

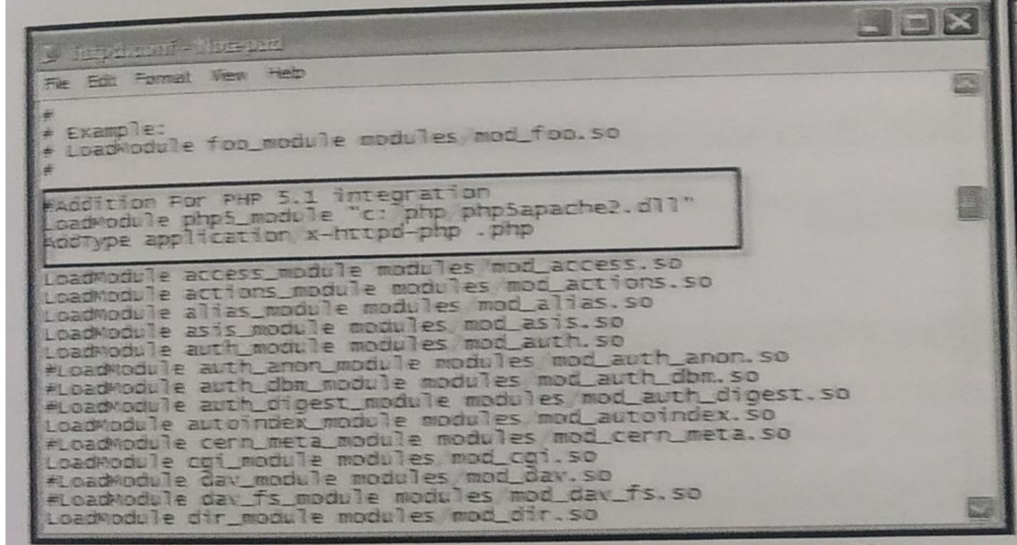
In the Apache 1.3.X, **AddModule** directive was used instead of **LoadModule**.

In case of Apache 1.3.X, search for the block AddModule statement. Add the following line after the last AddModule statement:

AddModule mod_php5.c

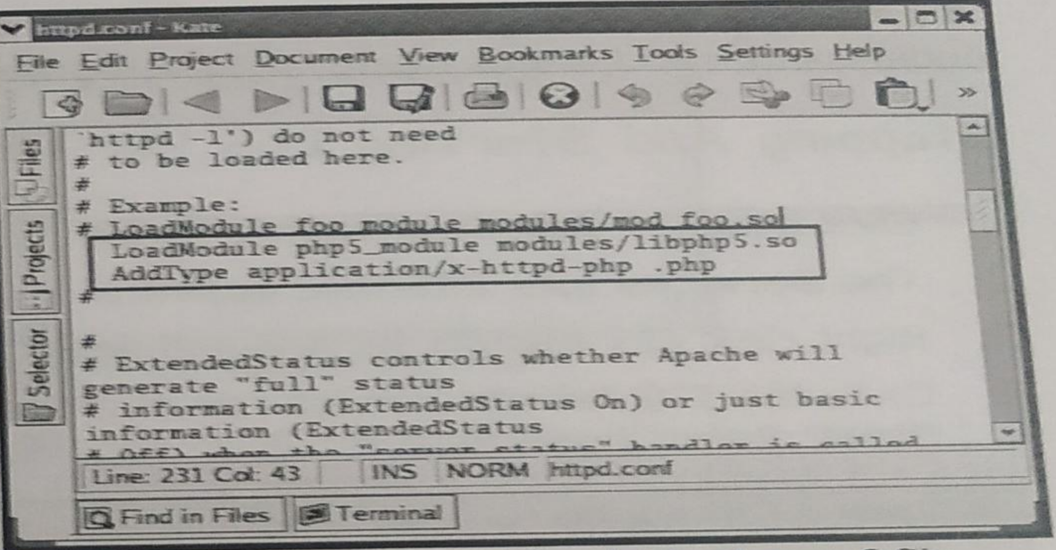
mod_php5.c file is not available anywhere in the file system. It only specifies the order in which the Apache Web server enables the various modules.

If Apache 2.X is used, **then do not insert** the AddModule directive. It's no longer needed in that version. Apache 2.X has its own internal method of determining the correct order of loading the modules.



```
#
# Example:
# LoadModule foo_module modules/mod_foo.so
#
#Addition For PHP 5.1 integration
LoadModule php5_module "c:/php/php5apache2.dll"
AddType application/x-httpd-php .php
LoadModule access_module modules/mod_access.so
LoadModule actions_module modules/mod_actions.so
LoadModule alias_module modules/mod_alias.so
LoadModule asis_module modules/mod_asis.so
LoadModule auth_module modules/mod_auth.so
#LoadModule auth_anon_module modules/mod_auth_anon.so
#LoadModule auth_dbm_module modules/mod_auth_dbm.so
#LoadModule auth_digest_module modules/mod_auth_digest.so
LoadModule autoindex_module modules/mod_autoindex.so
#LoadModule cern_meta_module modules/mod_cern_meta.so
LoadModule cgi_module modules/mod_cgi.so
#LoadModule dav_module modules/mod_dav.so
#LoadModule dav_fs_module modules/mod_dav_fs.so
LoadModule dir_module modules/mod_dir.so
```

Diagram 3.9.1: The **httpd.conf** file modification on Windows



```
`httpd -l') do not need
# to be loaded here.
#
# Example:
# LoadModule foo_module modules/mod_foo.so
 LoadModule php5_module modules/libphp5.so
 AddType application/x-httpd-php .php
#

#
# ExtendedStatus controls whether Apache will
generate "full" status
# information (ExtendedStatus On) or just basic
information (ExtendedStatus
```

Diagram 3.9.2: The **httpd.conf** file modification on Linux

Restart Apache server. PHP files should be able to be served up now.