PHP的命令行模式：

参考: <http://www.php100.com/cover/php/123.html>

# PHP 的命令行模式

从版本 4.3.0 开始，PHP 提供了一种新类型的 *SAPI*（Server Application Programming Interface，服务端应用编程端口）支持，名为 *CLI*，意为 Command Line Interface，即命令行接口。顾名思义，该 *SAPI* 模块主要用作 PHP 的开发外壳应用。*CLI SAPI* 和其它 *SAPI*模块相比有很多的不同之处，我们将在本章中详细阐述。值得一提的是，*CLI* 和 *CGI* 是不同的 SAPI，尽管它们之间有很多共同的行为。

*CLI SAPI* 最先是随 PHP 4.2.0 版本发布的，但仍旧只是一个实验性的版本，并需要在运行**./configure** 时加上 **--enable-cli** 参数(这句好像是编译吧)。从 PHP 4.3.0 版本开始，*CLI SAPI* 成为了正式模块，**--enable-cli** 参数会被默认得设置为 on，也可以用参数 **--disable-cli** 来屏蔽。

从 PHP 4.3.0开始，CLI/CGI 二进制执行文件的文件名、位置和是否存在会根据 PHP 在系统上的安装而不同。在默认情况下，当运行 **make** 时，CGI 和 CLI 都会被编译并且分别放置在 PHP 源文件目录的 sapi/cgi/php 和 sapi/cli/php 下。可以注意到两个文件都被命名为了 php。在 **make install** 的过程中会发生什么取决于配置行。如果在配置的时候选择了一个 SAPI 模块，如 apxs，或者使用了 **--disable-cgi** 参数，则在 **make install** 的过程中，CLI 将被拷贝到 {PREFIX}/bin/php，除非 CGI 已经被放置在了那个位置。因此，例如，如果在配置行中有 **--with--apxs** ，则在 *make install* 的过程中，CLI 将被拷贝到*{PREFIX}/bin/php*。如果希望撤销 CGI 执行文件的安装，请在 **make install** 之后运行 **make install-cli**。或者，也可以在配置行中加上 **--disable-cgi** 参数。

**Note**:

由于 **--enable-cli** 和 **--enable-cgi** 同时默认有效，因此，不必再配置行中加上 **--enable-cli** 来使得 CLI 在 **make install** 过程中被拷贝到 {PREFIX}/bin/php。

在 PHP 4.2.0 到 PHP 4.2.3 之间的 Windows 发行包中，CLI 的文件名为 php-cli.exe，相同文件夹下的 php.exe 为 CGI。从 PHP 4.3.0 版本开始，Windows 的发行包中 CLI 的执行文件为 php.exe，被放置在一个单独的名为 cli 的文件夹下，即 cli/php.exe。在 PHP 5 中，CLI 存在于主文件夹中，名为 php.exe，而 CGI 版本名为 php-cgi.exe。

从 PHP 5 起，一个名为 php-win.exe 的新文件随包发布。它相当于 CLI 版本，但是 php-win 不输出任何内容，便不提供控制台（不会弹出“DOS 窗口而且任务管理器没有识别”，cgi版本会弹出dos窗口并且显示在任务管理器中）。这种方式类似于 php-gtk。需要使用 **--enable-cli-win32** 选项来配置它。

**Note**: **如何得知自己使用的是哪个 SAPI？**

在命令行下，运行 **php -v** 便能得知该 php 是 CGI 还是 CLI。请参考函数[php\_sapi\_name()](http://www.php100.com/cover/php/1877.html) 以及常量 **PHP\_SAPI**。

**Note**:

在 PHP 4.3.2 中加入了 Unix 的 *man* 页面。可以在命令行中键入 **man php** 来查看。

以下为 *CLI SAPI* 和其它 *SAPI* 模块相比的显著区别：

区别：php-win.exe执行文件，什么也不输出

php.exe执行文件输出文内容，但不包含头信息

php-cgi.exe 执行文件内容输出信息，而且包括头信息。

* 与 *CGI SAPI* 不同，其输出没有任何头信息。

尽管 *CGI SAPI* 提供了取消 HTTP 头信息的方法，但在 *CLI SAPI* 中并不存在类似的方法以开启 HTTP 头信息的输出。

CLI 默认以安静模式开始，但为了保证兼容性，**-q** 和 **--no-header** 参数为了向后兼容仍然保留，使得可以使用旧的 CGI 脚本。

在运行时，不会把工作目录改为脚本的当前目录（可以使用 **-C** 和 **--no-chdir** 参数来兼容 CGI 模式）。

出错时输出纯文本的错误信息（非 HTML 格式）。

* *CLI SAPI* 强制覆盖了 php.ini 中的某些设置，因为这些设置在外壳环境下是没有意义的。

| **覆盖 php.ini 设置选项** | | |
| --- | --- | --- |
| **设置选项** | ***CLI SAPI* 默认值** | **备注** |
| html\_errors | **FALSE** | 无意义的 HTML 标记符会使得出错信息很凌乱，所以在外壳下阅读报错信息是十分困难的。因此将该选项的默认值改为 **FALSE**。 |
| implicit\_flush | **TRUE** | 在命令行模式下，所有来自 [print()](http://www.php100.com/cover/php/1931.html) 和 [echo()](http://www.php100.com/cover/php/415.html) 的输出将被立即写到输出端，而不作任何地缓冲操作。如果希望延缓或控制标准输出，仍然可以使用 [output buffering](http://www.php100.com/cover/php/2615.html) 设置项。 |
| max\_execution\_time | 0（无限值） | 鉴于在外壳环境下使用 PHP 的无穷的可能性，最大运行时间被设置为了无限值。为 web 开发的应用程序可能只需运行几秒钟时间，而外壳应用程序的运行时间可能会长的多。 |
| register\_argc\_argv | **TRUE** | 由于该设置为 **TRUE**，将总是可以在 *CLI SAPI* 中访问到 argc（传送给应用程序参数的个数）和 argv（包含有实际参数的数组）。  对于 PHP 4.3.0，在使用 *CLI SAPI* 时，PHP 变量*$argc* 和 *$argv* 已被注册并且设定了对应的值。而在这之前的版本，这两个变量在 *CGI* 或者 *模块* 版本中的建立依赖于将 PHP 的设置选项 register\_globals 设为 on。除了版本和*register\_globals* 设定以外，可以随时通过调用 $\_SERVER 或者 $HTTP\_SERVER\_VARS 来访问它们。例如：$\_SERVER['argv'] |

* **Note**:
* 这些设置无法在设置文件 php.ini 或任何指定的其它文件中被初始化为其它值。这些默认值被限制在所有其它的设置文件被解析后改变。不过，它们的值可以在程序运行的过程中被改变（尽管对于该运行过程来说，这些设置项是没有意义的）。
* 为了减轻外壳环境下的工作，我们定义了如下常量：

| **CLI 专用常量** | |
| --- | --- |
| **常量名称** | **描 述** |
| **STDIN** | 一个已打开的指向 *stdin* 的流。可以用如下方法来调用： <?php  $stdin = fopen('php://stdin', 'r');  ?> 如果想从 *stdin* 读取一行内容，可以使用 <?php $line = trim(fgets(STDIN)); // 从 STDIN 读取一行 fscanf(STDIN, "%d ", $number); // 从 STDIN 读取数字 ?> |
| **STDOUT** | 一个已打开的指向 *stdout* 的流。可以用如下方式来调用： <?php  $stdout = fopen('php://stdout', 'w');  ?> |
| **STDERR** | 一个已打开的指向 *stderr* 的流。可以用如下方式来调用： <?php  $stderr = fopen('php://stderr', 'w');  ?> |

* 有了以上常量，就无需自己建立指向诸如 *stderr* 的流，只需简单的使用这些常量来代替流指向：
* php -r 'fwrite(STDERR, "stderr
* ");'
* 无需自己来关闭这些流，PHP 会自动完成这些操作。
* *CLI SAPI* 不会将当前目录改为已运行的脚本所在的目录。

以下范例显示了本模块与 *CGI SAPI* 模块之间的不同： <?php  
// 名为 test.php 的简单测试程序  
echo getcwd(), " ";  
?>

在使用 *CGI* 版本时，其输出为

$ pwd

/tmp

$ php-cgi -f another\_directory/test.php

/tmp/another\_directory

明显可以看到 PHP 将当前目录改成了刚刚运行过的脚本所在的目录。

使用 *CLI SAPI* 模式，得到：

$ pwd

/tmp

$ php -q another\_directory/test.php

/tmp

这使得在利用 PHP 编写外壳工具时获得了很大的便利。

**Note**:

可以在命令行运行时给该 *CGI SAPI* 加上 **-C** 参数，使其支持 *CLI SAPI* 的功能。

以下是 PHP 二进制文件（即 php.exe 程序）提供的命令行模式的选项参数，随时可以运行带**-h** 参数的 PHP 命令来查询这些参数。

Usage: php [options] [-f] <file> [--] [args...]

php [options] -r <code> [--] [args...]

php [options] [-B <begin\_code>] -R <code> [-E <end\_code>] [--] [args...]

php [options] [-B <begin\_code>] -F <file> [-E <end\_code>] [--] [args...]

php [options] -- [args...]

php [options] -a

-a Run interactively

-c <path>|<file> Look for php.ini file in this directory

-n No php.ini file will be used

-d foo[=bar] Define INI entry foo with value 'bar'

-e Generate extended information for debugger/profiler

-f <file> Parse <file>.

-h This help

-i PHP information

-l Syntax check only (lint)

-m Show compiled in modules

-r <code> Run PHP <code> without using script tags <?..?>

-B <begin\_code> Run PHP <begin\_code> before processing input lines

-R <code> Run PHP <code> for every input line

-F <file> Parse and execute <file> for every input line

-E <end\_code> Run PHP <end\_code> after processing all input lines

-H Hide any passed arguments from external tools.

-s Display colour syntax highlighted source.

-v Version number

-w Display source with stripped comments and whitespace.

-z <file> Load Zend extension <file>.

args... Arguments passed to script. Use -- args when first argument

starts with - or script is read from stdin

*CLI SAPI* 模块有以下三种不同的方法来获取要运行的 PHP 代码：

1. 让 PHP 运行指定文件。
2. php my\_script.php #这是调用的是php.exe执行，输出内容不包括头信息
3. php -f my\_script.php

以上两种方法（使用或不使用 **-f** 参数）都能够运行给定的 my\_script.php 文件。可以选择任何文件来运行，指定的 PHP 脚本并非必须要以 *.php* 为扩展名，它们可以有任意的文件名和扩展名。

1. 在命令行直接运行 PHP 代码。
2. php -r 'print\_r(get\_defined\_constants());' #引号不是必须的，但是分号是必须的

在使用这种方法时，请注意外壳变量的替代及引号的使用。

**Note**:

请仔细阅读以上范例，在运行代码时没有开始和结束的标记符！加上 **-r** 参数后，这些标记符是不需要的，加上它们会导致语法错误。

1. 通过标准输入（*stdin*）提供需要运行的 PHP 代码。

以上用法提供了非常强大的功能，使得可以如下范例所示，动态地生成 PHP 代码并通过命令行运行这些代码：

$ some\_application | some\_filter | php | sort -u >final\_output.txt

以上三种运行代码的方法不能同时使用。

和所有的外壳应用程序一样，PHP 的二进制文件（php.exe 文件）及其运行的 PHP 脚本能够接受一系列的参数。PHP 没有限制传送给脚本程序的参数的个数（外壳程序对命令行的字符数有限制，但通常都不会超过该限制）。传递给脚本的参数可在全局变量 $argv 中获取。该数组中下标为零的成员为脚本的名称（当 PHP 代码来自标准输入获直接用 **-r** 参数以命令行方式运行时，该名称为“*-*”）。另外，全局变量 $argc （它是一个int型，代表参数的长度）存有 $argv 数组中成员变量的个数（而非传送给脚本程序的参数的个数）。

只要传送给脚本的参数不是以 *-* 符号开头，就无需过多的注意什么。向脚本传送以 *-* 开头的参数会导致错误，因为 PHP 会认为应该由它自身来处理这些参数。可以用参数列表分隔符 *--* 来解决这个问题。在 PHP 解析完参数后，该符号后所有的参数将会被原样传送给脚本程序。

# 以下命令将不会运行 PHP 代码，而只显示 PHP 命令行模式的使用说明：

$ php -r 'var\_dump($argv);' -h

Usage: php [options] [-f] <file> [args...]

[...]

# 以下命令将会把“-h”参数传送给脚本程序，PHP 不会显示命令行模式的使用说明：

$ php -r 'var\_dump($argv);' -- -h

array(2) {

[0]=>

string(1) "-"

[1]=>

string(2) "-h"

}

除此之外，还有另一个方法将 PHP 用于外壳脚本。可以在写一个脚本，并在第一行以*#!/usr/bin/php* 开头，在其后加上以 PHP 开始和结尾标记符包含的正常的 PHP 代码，然后为该文件设置正确的运行属性（例如：**chmod +x test**）。该方法可以使得该文件能够像外壳脚本或 PERL 脚本一样被直接执行。 #!/usr/bin/php  
<?php  
    var\_dump($argv);  
?>

假设改文件名为 test 并被放置在当前目录下，可以做如下操作：

$ chmod +x test

$ ./test -h -- foo

array(4) {

[0]=>

string(6) "./test"

[1]=>

string(2) "-h"

[2]=>

string(2) "--"

[3]=>

string(3) "foo"

}

正如所看到的，在向该脚本传送以 *-* 开头的参数时，脚本仍然能够正常运行。

PHP 4.3.3 以来有效的长选项:

| **命令行选项** | | |
| --- | --- | --- |
| **选项名称** | **长名称** | **说明** |
| -a | --interactive | 交互式运行 PHP。如果编译 PHP 时加入了 [Readline](http://www.php100.com/cover/php/2668.html) 扩展（Windows 下不可用），那将会得到一个很好的外壳，包括一个自动完成的功能（例如可以在键入变量名的时候，按下 TAB 键，PHP 会自动完成该变量名）以及命令历史记录，可以用上下键来访问。历史记录存在 ~/.php\_history 文件中。  **Note**:  通过 auto\_prepend\_file 和 auto\_append\_file 包含的文件在此模式下会被解析，但有些限制，例如函数必须在被调用之前定义。 |
| -c | --php-ini | 用该参数，可以指定一个放置 php.ini 文件的目录，或者直接指定一个自定义的 *INI* 文件（其文件名可以不是 php.ini），例如：  $ php -c /custom/directory/ my\_script.php  $ php -c /custom/directory/custom-file.ini my\_script.php  如果不指定此选项，PHP 将在默认位置搜索文件。 |
| -n | --no-php-ini | 完全忽略 php.ini。此参数在 PHP 4.3.0 以后有效。 |
| -d | --define | 用该参数可以自行设置任何可以在 php.ini 文件中设置的配置选项的值，其语法为：  -d configuration\_directive[=value]  例子（因版面原因而折行显示）：  # 取值部分被省略，将会把配置选项设为 "1"  $ php -d max\_execution\_time  -r '$foo = ini\_get("max\_execution\_time"); var\_dump($foo);'  string(1) "1"  # 取值部分为空白，将会把配置选项设为 ""  php -d max\_execution\_time=  -r '$foo = ini\_get("max\_execution\_time"); var\_dump($foo);'  string(0) ""  # 配置选项将被设置成为任何 '=' 字符之后的值  $ php -d max\_execution\_time=20  -r '$foo = ini\_get("max\_execution\_time"); var\_dump($foo);'  string(2) "20"  $ php  -d max\_execution\_time=doesntmakesense  -r '$foo = ini\_get("max\_execution\_time"); var\_dump($foo);'  string(15) "doesntmakesense" |
| -e | --profile-info | 激活扩展信息模式，被用于调试／测试。 |
| -f | --file | 解析并运行 **-f** 选项给定的文件名。该参数为可选参数，可以省略，仅指明需要运行的文件名即可。 |
| -h and -? | --help and --usage | 使用该参数，可以得到完整的命令行参数的列表及这些参数作用的简单描述。 |
| -i | --info | 该命令行参数会调用 phpinfo() 函数并显示出结果。如果 PHP 没有正常工作，建议执行 **php -i** 命令来查看在信息表格之前或者对应的地方是否有任何错误信息输出。请注意当使用 CGI 摸索时，输出的内容为 *HTML* 格式，因此输出的信息篇幅较大。 |
| -l | --syntax-check | 该参数提供了对指定 PHP 代码进行语法检查的方便的方法。如果成功，则向标准输出写入 *No syntax errors detected in <filename>* 字符串，并且外壳返回值为 *0*。如果失败，则输出 *Errors parsing <filename>* 以及内部解析器错误信息到标准输出，同时外壳返回值将别设置为 *255*。  该参数将无法检查致命错误（如未定义函数），如果也希望检测致命错误，请使用 **-f** 参数。  **Note**:  该参数不能和 **-r** 一同使用。 |
| -m | --modules | 使用该参数，PHP 将打印出内置以及已加载的 PHP 及 Zend 模块：  $ php -m  [PHP Modules]  xml  tokenizer  standard  session  posix  pcre  overload  mysql  mbstring  ctype  [Zend Modules] |
| -r | --run | 使用该参数可以在命令行内运行单行 PHP 代码。无需加上 PHP 的起始和结束标识符（*<?php* 和 *?>*），否则将会导致语法解析错误。  **Note**:  使用这种形式的 PHP 时，应注意避免和外壳环境进行的命令行参数替换相冲突。  显示语法解析错误的范例  $ php -r "$foo = get\_defined\_constants();"  Command line code(1) : Parse error - parse error, unexpected '='  这里的问题在于即使使用了双引号 *"*，sh/bash 仍然实行了参数替换。由于 $foo 没有被定义，被替换后它所在的位置变成了空字符，因此在运行时，实际被 PHP 读取的代码为：  $ php -r " = get\_defined\_constants();"  正确的方法是使用单引号 *'*。在用单引号引用的字符串中，变量不会被 sh/bash 还原成其原值。  $ php -r '$foo = get\_defined\_constants(); var\_dump($foo);'  array(370) {  ["E\_ERROR"]=>  int(1)  ["E\_WARNING"]=>  int(2)  ["E\_PARSE"]=>  int(4)  ["E\_NOTICE"]=>  int(8)  ["E\_CORE\_ERROR"]=>  [...]  如果使用的外壳不是 sh/bash，可能会碰到更多问题。请将碰到的 Bug 向 [» http://bugs.php.net/](http://www.php100.com/cover/php/bugs.php.net/index.html) 报告。注意，当试图将 shell 变量用到代码中或者使用反斜线时仍然很容易碰到问题。  **Note**:  **-r** 在 CLI SAPI 中有效，在 CGI SAPI 中无效。  **Note**:  此选项只用于非常基本的用途。因此一些配置指令（例如 auto\_prepend\_file 和 auto\_append\_file）在此模式下被忽略。 |
| -B | --process-begin | 在处理 stdin 之前先执行 PHP 代码。PHP 5 新加。 |
| -R | --process-code | 对每个输入行都执行 PHP 代码。PHP 5 新加。  此模式下有两个特殊变量：$argn 和 $argi。$argn 包含 PHP 当前处理的行内容，而$argi 则包含该行号。 |
| -F | --process-file | 对每个输入行都执行 PHP 文件。PHP 5 新加。 |
| -E | --process-end | 在处理完输入后执行的 PHP 代码。PHP 5 新加。  使用 **-B** ，**-R** 和 **-E** 选项来计算一个项目总行数的例子。  $ find my\_proj | php -B '$l=0;' -R '$l += count(@file($argn));' -E 'echo "Total Lines: $l  ";'  Total Lines: 37328 |
| -s | --syntax-highlight and --syntax-highlight | 显示有语法高亮色彩的源代码。  该参数使用内建机制来解析文件并为其生成一个 *HTML* 高亮版本并将结果写到标准输出。请注意该过程所做的只是生成了一个 *<code> [...] </code>* 的 *HTML* 标记的块，并不包含任何的 *HTML* 头。  **Note**:  该选项不能和 **-r** 参数同时使用。 |
| -v | --version | 将 PHP，PHP SAPI 和 Zend 的版本信息写入标准输出。例如：  $ php -v  PHP 4.3.0 (cli), Copyright (c) 1997-2002 The PHP Group  Zend Engine v1.3.0, Copyright (c) 1998-2002 Zend Technologies |
| -w | --strip | 显示除去了注释和多余空白的源代码。  **Note**:  该选项不能和 **-r** 参数同时使用。 |
| -z | --zend-extension | 加载 Zend 扩展库。如果仅给定一个文件名，PHP 将试图从当前系统扩展库的默认路径（在 Linux 系统下，该路径通常由 /etc/ld.so.conf 指定）加载该扩展库。如果用一个绝对路径指定文件名，则不会使用系统的扩展库默认路径。如果用相对路径指定的文件名，则 PHP 仅试图在当前目录的相对目录加载扩展库。 |

PHP 的命令行模式能使得 PHP 脚本能完全独立于 web 服务器单独运行。如果使用 Unix 系统，需要在 PHP 脚本的最前面加上一行特殊的代码，使得它能够被执行，这样系统就能知道用哪个程序去运行该脚本。在 Windows 平台下可以将 php.exe 和 *.php* 文件的双击属性相关联，也可以编写一个批处理文件来用 PHP 执行脚本。为 Unix 系统增加的第一行代码不会影响该脚本在 Windows 下的运行，因此也可以用该方法编写跨平台的脚本程序。以下是一个简单的 PHP 命令行程序的范例。

**Example #1 试图以命令行方式运行的 PHP 脚本（script.php）**

#!/usr/bin/php  
<?php  
  
if ($argc != 2 || in\_array($argv[1], array('--help', '-help', '-h', '-?'))) {  
?>  
  
This is a command line PHP script with one option.  
  
  Usage:  
  <?php echo $argv[0]; ?> <option>  
  
  <option> can be some word you would like  
  to print out. With the --help, -help, -h,  
  or -? options, you can get this help.  
  
<?php  
} else {  
    echo $argv[1];  
}  
?>

在以上脚本中，用第一行特殊的代码来指明该文件应该由 PHP 来执行。在这里使用 CLI 的版本，因此不会有 HTTP 头信息输出。在用 PHP 编写命令行应用程序时，可以使用两个参数：$argc 和 $argv。前面一个的值是比参数个数大 1 的整数（运行的脚本本身的名称也被当作一个参数）。第二个是包含有参数的数组，其第一个元素为脚本的名称，下标为数字 0（$argv[0]）。

以上程序中检查了参数的个数是大于 1 个还是小于 1 个。此外如果参数是 **--help** ，**-help**，**-h** 或 **-?** 时，打印出帮助信息，并同时动态输出脚本的名称。如果还收到了其它参数，将其显示出来。

如果希望在 Unix 下运行以上脚本，需要使其属性为可执行文件，然后简单的运行 **script.php echothis** 或 **script.php -h**。在 Windows 下，可以为此编写一个批处理文件：

**Example #2 运行 PHP 命令行脚本的批处理文件（script.bat）**

@C:phpphp.exe script.php %1 %2 %3 %

假设将上述程序命名为 script.php，且 CLI 版的 php.exe 文件放置在 c:phpcliphp.exe，该批处理文件会帮助将附加的参数传给脚本程序：**script.bat echothis** 或 **script.bat -h**。

请参阅 [Readline](http://www.php100.com/cover/php/2668.html) 扩展模块的有关文档，以获取更多的函数的信息。这些函数可以帮助完善 PHP 命令行应用程序。