Change language: English

Using PHP from the command line

Table of Contents

- Differences to other SAPIs
- Options
- Usage
- I/O streams
- Interactive shell
- Built-in web server
- INI settings

Introduction

The main focus of CLI SAPI is for developing shell applications with PHP. There are quite a few differences between the CLI SAPI and other SAPIs which are explained in this chapter. It is worth mentioning that CLI and CGI are different SAPIs although they do share many of the same behaviors.

The CLI SAPI is enabled by default using **--enable-cli**, but may be disabled using the **--disable-cli** option when running **./configure**.

The name, location and existence of the CLI/CGI binaries will differ depending on how PHP is installed on your system. By default when executing **make**, both the CGI and CLI are built and placed as `sapi/cgi/php-cgi` and `sapi/cli/php` respectively, in your PHP source directory. You will note that both are named `php`. What happens during **make install** depends on your configure line. If a module SAPI is chosen during configure, such as apxs, or the **--disable-cgi** option is used, the CLI is copied to `{PREFIX}/bin/php` during **make install** otherwise the CGI is placed there. So, for example, if **--with-apxs** is in your configure line then the CLI is copied to `{PREFIX}/bin/php` during **make install**. If you want to override the installation of the CGI binary, use **make install-cli** after **make install**. Alternatively you can specify **--disable-cgi** in your configure line.

> **Note**:
> Because both **--enable-cli** and **--enable-cgi** are enabled by default, simply having **--enable-cli** in your configure line does not necessarily mean the CLI will be copied as `{PREFIX}/bin/php` during **make install**.

The CLI binary is distributed in the main folder as `php.exe` on Windows. The CGI version is distributed as `php-cgi.exe`. Additionally, a `php-win.exe` is distributed if PHP is configured using **--enable-cli-win32**. This does the same as the CLI version, except that it doesn't output anything and thus provides no console.

> **Note**: **What SAPI do I have?**
> From a shell, typing **php -v** will tell you whether `php` is CGI or CLI. See also the function php_sapi_name() and the constant **PHP_SAPI**.

> **Note**:
> A Unix `manual` page is available by typing **man php** in the shell environment.

Features
- HTTP authentication with PHP
- Cookies
- Sessions
- Handling file uploads
- Using remote files
- Connection handling
- Persistent Database Connections
- » Command line usage
- Garbage Collection
- DTrace Dynamic Tracing

Improve This Page

Learn How To Improve This Page • Submit a Pull Request • Report a Bug

User Contributed Notes 33 notes

+ add a note

109 sep16 at psu dot edu 12 years ago

```
You can easily parse command line arguments into the $_GET variable by using the
parse_str() function.

<?php

parse_str(implode('&', array_slice($argv, 1)), $_GET);

?>

It behaves exactly like you'd expect with cgi-php.

$ php -f somefile.php a=1 b[]=2 b[]=3

This will set $_GET['a'] to '1' and $_GET['b'] to array('2', '3').

Even better, instead of putting that line in every file, take advantage of PHP's
auto_prepend_file directive.  Put that line in its own file and set the auto_prepend_file
directive in your cli-specific php.ini like so:

auto_prepend_file = "/etc/php/cli-php5.3/local.prepend.php"

It will be automatically prepended to any PHP file run from the command line.
```

22 apmuthu at usa dot net 6 years ago

```
Adding a pause() function to PHP waiting for any user input returning it:

<?php
function pause() {
    $handle = fopen ("php://stdin","r");
    do { $line = fgets($handle); } while ($line == '');
    fclose($handle);
    return $line;
}
?>
```

23 frankNospamwanted at. toppoint dot. de 10 years ago

```
Parsing commandline argument GET String without changing the PHP script (linux shell):
URL: index.php?a=1&b=2
Result: output.html

echo "" | php -R 'include("index.php");' -B 'parse_str($argv[1], $_GET);' 'a=1&b=2'
>output.html

(no need to change php.ini)

You can put this
  echo "" | php -R 'include("'$1'");' -B 'parse_str($argv[1], $_GET);' "$2"
in a bash script "php_get" to use it like this:
  php_get index.php 'a=1&b=2' >output.html
```

```
or directed to text browser...
  php_get index.php 'a=1&b=2' |w3m -T text/html
```

16 PSIKYO at mail dot dlut dot edu dot cn — 11 years ago

```
If you edit a php file in windows, upload and run it on linux with command line method.
You may encounter a running problem probably like that:

[root@ItsCloud02 wsdl]# ./lnxcli.php
Extension './lnxcli.php' not present.

Or you may encounter some other strange problem.
Care the enter key. In windows environment, enter key generate two binary characters
'0D0A'. But in Linux, enter key generate just only a 'OA'.
I wish it can help someone if you are using windows to code php and run it as a command
line program on linux.
```

15 ohcc at 163 dot com — 8 years ago

```
use " instead of ' on windows when using the cli version with -r

php -r "echo 1"
-- correct

php -r 'echo 1'
  PHP Parse error:  syntax error, unexpected ''echo' (T_ENCAPSED_AND_WHITESPACE),
expecting end of file in Command line code on line 1
```

18 drewish at katherinehouse dot com — 19 years ago

```
When you're writing one line php scripts remember that 'php://stdin' is your friend.
Here's a simple program I use to format PHP code for inclusion on my blog:

UNIX:
  cat test.php | php -r "print htmlentities(file_get_contents('php://stdin'));"

DOS/Windows:
  type test.php | php -r "print htmlentities(file_get_contents('php://stdin'));"
```

21 Kodeart — 13 years ago

```
Check directly without calling functions:

<?php

if (PHP_SAPI === 'cli')

{

   // ...

}

?>



You can define a constant to use it elsewhere

<?php

define('ISCLI', PHP_SAPI === 'cli');
```

```
?>
```

16 monte at ispi dot net 21 years ago

I had a problem with the $argv values getting split up when they contained plus (+) signs. Be sure to use the CLI version, not CGI to get around it.

Monte

14 lucas dot vasconcelos at gmail dot com 17 years ago

Just another variant of previous script that group arguments doesn't starts with '-' or '--'

```
<?php

function arguments($argv) {

    $_ARG = array();

    foreach ($argv as $arg) {

      if (ereg('--([^=]+)=(.*)',$arg,$reg)) {

        $_ARG[$reg[1]] = $reg[2];

      } elseif(ereg('^-([a-zA-Z0-9])',$arg,$reg)) {

            $_ARG[$reg[1]] = 'true';

      } else {

            $_ARG['input'][]=$arg;

      }

    }

  return $_ARG;

}



print_r(arguments($argv));

?>



$ php myscript.php --user=nobody /etc/apache2/*

Array

(

    [input] => Array

        (
            [0] => myscript.php

            [1] => /etc/apache2/apache2.conf

            [2] => /etc/apache2/conf.d

            [3] => /etc/apache2/envvars

            [4] => /etc/apache2/httpd.conf

            [5] => /etc/apache2/mods-available

            [6] => /etc/apache2/mods-enabled

            [7] => /etc/apache2/ports.conf

            [8] => /etc/apache2/sites-available

            [9] => /etc/apache2/sites-enabled

        )



    [user] => nobody

)
```

15 Anonymous 3 years ago

```
We can pass many arguments directly into the hashbang line.
As example many ini setting via the -d parameter of php.
---
#!/usr/bin/php -d memory_limit=2048M -d post_max_size=0
phpinfo();
exit;
---
./script | grep memory
memory_limit => 2048M => 2048M
---
But we can also use this behaviour into a second script, so it call the first as an
interpreter, via the hashbang:
---
#!./script arg1 arg2 arg3
---
However the parameters are dispatched in a different way into $argv

All the parameters are in $argv[1], $argv[0] is the interpreter script name, and $argv[1]
is the caller script name.

To get back the parameters into $argv, we can simply test if $argv[1] contains spaces,
and then dispatch again as normal:

#!/usr/bin/php -d memory_limit=2048M -d post_max_size=0
<?php
var_dump($argv);
if (strpos($argv[1], ' ') !== false){
  $argw = explode(" ", $argv[1]);
  array_unshift($argw, $argv[2]);
  $argv = $argw;
}
```

```
var_dump($argv); ?>
---
array(3) {
  [0]=>
  string(8) "./script"
  [1]=>
  string(15) "arg1 arg2 arg3 "
  [2]=>
  string(14) "./other_script"
}
array(4) {
  [0]=>
  string(8) "./other_script"
  [1]=>
  string(4) "arg1"
  [2]=>
  string(4) "arg2"
  [3]=>
  string(4) "arg3"
}
---
This will maintain the same behaviour in all cases and allow to even double click a
script to call both parameters of another script, and even make a full interpreter
language layer.  The other script doesn't has to be php. Take care of paths.
```

26 ben at slax0rnet dot com 20 years ago

```
Just a note for people trying to use interactive mode from the commandline.

The purpose of interactive mode is to parse code snippits without actually leaving php,
and it works like this:

[root@localhost php-4.3.4]# php -a
Interactive mode enabled

<?php echo "hi!"; ?>
<note, here we would press CTRL-D to parse everything we've entered so far>
hi!
<?php exit(); ?>
<ctrl-d here again>
[root@localhost php-4.3.4]#

I noticed this somehow got ommited from the docs, hope it helps someone!
```

20 notreallyanaddress at somerandomaddr dot com 14 years ago

```
If you want to be interactive with the user and accept user input, all you need to do is
read from stdin.

<?php
echo "Are you sure you want to do this?  Type 'yes' to continue: ";
$handle = fopen ("php://stdin","r");
$line = fgets($handle);
if(trim($line) != 'yes'){
    echo "ABORTING!\n";
    exit;
}
echo "\n";
echo "Thank you, continuing...\n";
?>
```

15 OverFlow636 at gmail dot com 19 years ago

I needed this, you proly wont tho.

puts the exicution args into $_GET

```
<?php

if ($argv) {

    foreach ($argv as $k=>$v)

    {

        if ($k==0) continue;

        $it = explode("=",$argv[$i]);

        if (isset($it[1])) $_GET[$it[0]] = $it[1];

    }

}

?>
```

16 thomas dot harding at laposte dot net 16 years ago

Parsing command line: optimization is evil!

One thing all contributors on this page forgotten is that you can suround an argv with single or double quotes. So the join coupled together with the preg_match_all will always break that :)

Here is a proposal:

```
#!/usr/bin/php
<?php
print_r(arguments($argv));

function arguments ( $args )
{
  array_shift( $args );
  $endofoptions = false;

  $ret = array
    (
    'commands' => array(),
    'options' => array(),
    'flags'    => array(),
    'arguments' => array(),
    );

  while ( $arg = array_shift($args) )
  {

    // if we have reached end of options,
    //we cast all remaining argvs as arguments
    if ($endofoptions)
    {
      $ret['arguments'][] = $arg;
      continue;
    }

    // Is it a command? (prefixed with --)
    if ( substr( $arg, 0, 2 ) === '--' )
    {

      // is it the end of options flag?
      if (!isset ($arg[3]))
      {
        $endofoptions = true;; // end of options;
        continue;
      }

      $value = "";
      $com   = substr( $arg, 2 );

      // is it the syntax '--option=argument'?
      if (strpos($com,'='))
        list($com,$value) = split("=",$com,2);

      // is the option not followed by another option but by arguments
      elseif (strpos($args[0],'-') !== 0)
      {
        while (strpos($args[0],'-') !== 0)
          $value .= array_shift($args).' ';
        $value = rtrim($value,' ');
      }

      $ret['options'][$com] = !empty($value) ? $value : true;
      continue;

    }

    // Is it a flag or a serial of flags? (prefixed with -)
    if ( substr( $arg, 0, 1 ) === '-' )
    {
      for ($i = 1; isset($arg[$i]) ; $i++)
        $ret['flags'][] = $arg[$i];
      continue;
    }

    // finally, it is not option, nor flag, nor argument
    $ret['commands'][] = $arg;
    continue;
  }

  if (!count($ret['options']) && !count($ret['flags']))
  {
    $ret['arguments'] = array_merge($ret['commands'], $ret['arguments']);
    $ret['commands'] = array();
  }
return $ret;
}

exit (0)

/* vim: set expandtab tabstop=2 shiftwidth=2: */
?>
```

▲ 11 ▼ rob 17 years ago

```
i use emacs in c-mode for editing.  in 4.3, starting a cli script like so:

#!/usr/bin/php -q /* -*- c -*- */
<?php

told emacs to drop into c-mode automatically when i loaded the file for editing.  the '-
q' flag didn't actually do anything (in the older cgi versions, it suppressed html output
when the script was run) but it caused the commented mode line to be ignored by php.

in 5.2, '-q' has apparently been deprecated.  replace it with '--' to achieve the 4.3
invocation-with-emacs-mode-line behavior:

#!/usr/bin/php -- /* -*- c -*- */
<?php

don't go back to your 4.3 system and replace '-q' with '--'; it seems to cause php to
hang waiting on STDIN...
```

▲ 11 ▼ goalain eat gmail dont com — 17 years ago

```
If your php script doesn't run with shebang (#!/usr/bin/php),

and it issues the beautifull and informative error message:

"Command not found."  just dos2unix yourscript.php

et voila.



If you still get the "Command not found."

Just try to run it as ./myscript.php , with the "./"

if it works - it means your current directory is not in the executable search path.



If your php script doesn't run with shebang (#/usr/bin/php),

and it issues the beautifull and informative message:

"Invalid null command." it's probably because the "!" is missing in the the shebang line
(like what's above) or something else in that area.



\Alon
```

▲ 9 ▼ jeff at noSpam[] dot genhex dot net — 22 years ago

```
You can also call the script from the command line after chmod'ing the file (ie: chmod
755 file.php).

On your first line of the file, enter "#!/usr/bin/php" (or to wherever your php
executable is located).  If you want to suppress the PHP headers, use the line of
"#!/usr/bin/php -q" for your path.
```

▲ 8 ▼ roberto dot dimas at gmail dot com — 19 years ago

```
One of the things I like about perl and vbscripts, is the fact that I can name a file
e.g. 'test.pl' and just have to type 'test, without the .pl extension' on the windows
command line and the command processor knows that it is a perl file and executes it using
the perl command interpreter.

I did the same with the file extension .php3 (I will use php3 exclusivelly for command
line php scripts, I'm doing this because my text editor VIM 6.3 already has the correct
syntax highlighting for .php3 files ).

I modified the PATHEXT environment variable in Windows XP, from the " 'system' control
panel applet->'Advanced' tab->'Environment Variables' button-> 'System variables' text
area".

Then from control panel "Folder Options" applet-> 'File Types' tab, I added a new file
extention (php3), using the button 'New'  and typing php3 in the window that pops up.

Then in the 'Details for php3 extention' area I used the 'Change' button to look for the
Php.exe executable so that the php3 file extentions are associated with the php
executable.

You have to modify also the 'PATH' environment variable, pointing to the folder where the
php executable is installed

Hope this is useful to somebody
```

▲ 7 ▼ phpnotes at ssilk dot de 22 years ago

```
To hand over the GET-variables in interactive mode like in HTTP-Mode (e.g. your URI is
myprog.html?hugo=bla&bla=hugo), you have to call

php myprog.html '&hugo=bla&bla=hugo'

(two & instead of ? and &!)

There just a little difference in the $ARGC, $ARGV values, but I think this is in those
cases not relevant.
```

▲ 8 ▼ Anonymous 14 years ago

```
Using CLI (on WIN at least), some INI paths are relative to the current working
directory.  For example, if your error_log = "php_errors.log", then php_errors.log will
be created (or appended to if already exists) in whatever directory you happen to be in
at the moment if you have write access there.  Instead of having random error logs all
over the place because of this behavior, you may want to set error_log to a full path,
perhaps to the php.exe directory.
```

▲ 9 ▼ linn at backendmedia dot com 20 years ago

```
For those of you who want the old CGI behaviour that changes to the actual directory of
the script use:
chdir(dirname($_SERVER['argv'][0]));

at the beginning of your scripts.
```

▲ 9 ▼ eric dot brison at anakeen dot com 17 years ago

```
Just a variant of previous script to accept arguments with '=' also
<?php
function arguments($argv) {
    $_ARG = array();
    foreach ($argv as $arg) {
      if (ereg('--([^=]+)=(.*)',$arg,$reg)) {
        $_ARG[$reg[1]] = $reg[2];
      } elseif(ereg('-([a-zA-Z0-9])',$arg,$reg)) {
            $_ARG[$reg[1]] = 'true';
        }

    }
  return $_ARG;
}
?>
$ php myscript.php --user=nobody --password=secret -p --access="host=127.0.0.1 port=456"
Array
(
    [user] => nobody
    [password] => secret
    [p] => true
    [access] => host=127.0.0.1 port=456
)
```

▲ 4 ▼ sam marshall 5 years ago

```
When using the -R flag, the name of the variable containing the content of the current
line (not including the LF) is $argn.

For example you can do this code:

cat file.txt | php -R 'echo $argn . "\n";'

This will just output each line of the input file without doing anything to it.
```

▲ 6 ▼ losbrutos at free dot fr 17 years ago

```
an another "another variant" :

<?php
function arguments($argv)
{
  $_ARG = array();
  foreach ($argv as $arg)
  {
    if (preg_match('#^-{1,2}([a-zA-Z0-9]*)=?(.*)$#', $arg, $matches))
    {
      $key = $matches[1];
      switch ($matches[2])
      {
        case '':
        case 'true':
          $arg = true;
          break;
        case 'false':
          $arg = false;
          break;
        default:
          $arg = $matches[2];
      }
      $_ARG[$key] = $arg;
    }
    else
    {
      $_ARG['input'][] = $arg;
    }
  }
  return $_ARG;
}
?>

$php myscript.php arg1 -arg2=val2 --arg3=arg3 -arg4 --arg5 -arg6=false

Array
(
    [input] => Array
        (
            [0] => myscript.php
            [1] => arg1
        )

    [arg2] => val2
    [arg3] => arg3
    [arg4] => true
    [arg5] => true
    [arg5] => false
)
```

7 docey 19 years ago

```
dunno if this is on linux the same but on windows evertime
you send somthing to the console screen php is waiting for
the console to return. therefor if you send a lot of small
short amounts of text, the console is starting to be using
more cpu-cycles then php and thus slowing the script.

take a look at this sheme:
cpu-cycle:1 ->php: print("a");
cpu-cycle:2 ->cmd: output("a");
cpu-cycle:3 ->php: print("b");
cpu-cycle:4 ->cmd: output("b");
cpu-cycle:5 ->php: print("c");
cpu-cycle:6 ->cmd: output("c");
cpu-cylce:7 ->php: print("d");
cpu-cycle:8 ->cmd: output("d");
cpu-cylce:9 ->php: print("e");
cpu-cycle:0 ->cmd: output("e");

on the screen just appears "abcde". but if you write
your script this way it will be far more faster:
cpu-cycle:1 ->php: ob_start();
cpu-cycle:2 ->php: print("abc");
cpu-cycle:3 ->php: print("de");
cpu-cycle:4 ->php: $data = ob_get_contents();
cpu-cycle:5 ->php: ob_end_clean();
cpu-cycle:6 ->php: print($data);
cpu-cycle:7 ->cmd: output("abcde");

now this is just a small example but if you are writing an
app that is outputting a lot to the console, i.e. a text
based screen with frequent updates, then its much better
to first cach all output, and output is as one big chunk of
text instead of one char a the time.

ouput buffering is ideal for this. in my script i outputted
almost 4000chars of info and just by caching it first, it
speeded up by almost 400% and dropped cpu-usage.

because what is being displayed doesn't matter, be it 2
```

```
chars or 40.0000 chars, just the call to output takes a
great deal of time. remeber that.

maybe someone can test if this is the same on unix-based
systems. it seems that the STDOUT stream just waits for
the console to report ready, before continueing execution.
```

8 Adam, php(at)getwebspace.com 21 years ago

```
Ok, I've had a heck of a time with PHP > 4.3.x and whether to use CLI vs CGI. The CGI
version of 4.3.2 would return (in browser):
---
No input file specified.
---

And the CLI version would return:
---
500 Internal Server Error
---

It appears that in CGI mode, PHP looks at the environment variable PATH_TRANSLATED to
determine the script to execute and ignores command line. That is why in the absensce of
this environment variable, you get "No input file specified." However, in CLI mode the
HTTP headers are not printed. I believe this is intended behavior for both situations but
creates a problem when you have a CGI wrapper that sends environment variables but passes
the actual script name on the command line.

By modifying my CGI wrapper to create this PATH_TRANSLATED environment variable, it
solved my problem, and I was able to run the CGI build of 4.3.2
```

5 obfuscated at emailaddress dot com 19 years ago

```
This posting is not a php-only problem, but hopefully will save someone a few hours of
headaches.  Running on MacOS (although this could happen on any *nix I suppose), I was
unable to get the script to execute without specifically envoking php from the command
line:

[macg4:valencia/jobs] tim% test.php
./test.php: Command not found.

However, it worked just fine when php was envoked on the command line:

[macg4:valencia/jobs] tim% php test.php
Well, here we are...  Now what?

Was file access mode set for executable?  Yup.

[macg4:valencia/jobs] tim% ls -l
total 16
-rwxr-xr-x  1 tim  staff   242 Feb 24 17:23 test.php

And you did, of course, remember to add the php command as the first line of your script,
yeah?  Of course.

#!/usr/bin/php
<?php print "Well, here we are...  Now what?\n"; ?>

So why dudn't it work?  Well, like I said... on a Mac.... but I also occasionally edit
the files on my Windows portable (i.e. when I'm travelling and don't have my trusty Mac
available)...  Using, say, WordPad on Windows... and BBEdit on the Mac...

Aaahhh... in BBEdit check how the file is being saved!  Mac?  Unix?  or Dos?  Bingo.  It
```

```
had been saved as Dos format.  Change it to Unix:

[macg4:valencia/jobs] tim% ./test.php
Well, here we are...  Now what?
[macg4:valencia/jobs] tim%

NB: If you're editing your php files on multiple platforms (i.e. Windows and Linux), make
sure you double check the files are saved in a Unix format...  those \r's and \n's 'll
bite cha!
```

▲ 4 ▼ bluej100@gmail 17 years ago

```
In 5.1.2 (and others, I assume), the -f form silently drops the first argument after the
script name from $_SERVER['argv']. I'd suggest avoiding it unless you need it for a
special case.
```

▲ 4 ▼ Popeye at P-t-B dot com 21 years ago

```
In *nix systems, use the WHICH command to show the location of the php binary executable.
This is the path to use as the first line in your php shell script file. (#!/path/to/php
-q) And execute php from the command line with the -v switch to see what version you are
running.

example:

# which php
/usr/local/bin/php
# php -v
PHP 4.3.1 (cli) (built: Mar 27 2003 14:41:51)
Copyright (c) 1997-2002 The PHP Group
Zend Engine v1.3.0, Copyright (c) 1998-2002 Zend Technologies

In the above example, you would use: #!/usr/local/bin/php

Also note that, if you do not have the current/default directory in your PATH (.), you
will have to use ./scriptfilename to execute your script file from the command line (or
you will receive a "command not found" error). Use the ENV command to show your PATH
environment variable value.
```

▲ 4 ▼ Alexander Plakidin 21 years ago

```
How to change current directory in PHP script to script's directory when running it from
command line using PHP 4.3.0?
(you'll probably need to add this to older scripts when running them under PHP 4.3.0 for
backwards compatibility)

Here's what I am using:
chdir(preg_replace('/\\/[^\\/]+$/',"",$PHP_SELF));

Note: documentation says that "PHP_SELF" is not available in command-line PHP scripts.
Though, it IS available. Probably this will be changed in future version, so don't rely
on this line of code...

Use $_SERVER['PHP_SELF'] instead of just $PHP_SELF if you have register_globals=Off
```

▲ 4 ▼ stromdotcom at hotmail dot com 18 years ago

```
Spawning php-win.exe as a child process to handle scripting in Windows applications has a
few quirks (all having to do with pipes between Windows apps and console apps).

To do this in C++:
```

```
// We will run php.exe as a child process after creating
// two pipes and attaching them to stdin and stdout
// of the child process
// Define sa struct such that child inherits our handles

SECURITY_ATTRIBUTES sa = { sizeof(SECURITY_ATTRIBUTES) };
sa.bInheritHandle = TRUE;
sa.lpSecurityDescriptor = NULL;

// Create the handles for our two pipes (two handles per pipe, one for each end)
// We will have one pipe for stdin, and one for stdout, each with a READ and WRITE end
HANDLE hStdoutRd, hStdoutWr, hStdinRd, hStdinWr;

// Now create the pipes, and make them inheritable
CreatePipe (&hStdoutRd, &hStdoutWr, &sa, 0))
SetHandleInformation(hStdoutRd, HANDLE_FLAG_INHERIT, 0);
CreatePipe (&hStdinRd, &hStdinWr, &sa, 0)
SetHandleInformation(hStdinWr, HANDLE_FLAG_INHERIT, 0);

// Now we have two pipes, we can create the process
// First, fill out the usage structs
STARTUPINFO si = { sizeof(STARTUPINFO) };
PROCESS_INFORMATION pi;
si.dwFlags = STARTF_USESTDHANDLES;
si.hStdOutput = hStdoutWr;
si.hStdInput  = hStdinRd;

// And finally, create the process
CreateProcess (NULL, "c:\\php\\php-win.exe", NULL, NULL, TRUE, NORMAL_PRIORITY_CLASS,
NULL, NULL, &si, &pi);

// Close the handles we aren't using
CloseHandle(hStdoutWr);
CloseHandle(hStdinRd);

// Now that we have the process running, we can start pushing PHP at it
WriteFile(hStdinWr, "<?php echo 'test'; ?>", 9, &dwWritten, NULL);

// When we're done writing to stdin, we close that pipe
CloseHandle(hStdinWr);

// Reading from stdout is only slightly more complicated
int i;

std::string processed("");
char buf[128];

while ( (ReadFile(hStdoutRd, buf, 128, &dwRead, NULL) && (dwRead != 0)) ) {
    for (i = 0; i < dwRead; i++)
        processed += buf[i];
}

// Done reading, so close this handle too
CloseHandle(hStdoutRd);

A full implementation (implemented as a C++ class) is available at
http://www.stromcode.com
```

▲ 3 ▼ pyxl at jerrell dot com 22 years ago

```
Assuming --prefix=/usr/local/php, it's better to create a symlink from /usr/bin/php or
/usr/local/bin/php to target /usr/local/php/bin/php so that it's both in your path and
```

```
automatically correct every time you rebuild.  If you forgot to do that copy of the
binary after a rebuild, you can do all kinds of wild goose chasing when things break.
```

▲ 2 ▼ james_s2010 at NOSPAM dot hotmail dot com 17 years ago

I was looking for a way to interactively get a single character response from user. Using STDIN with fread, fgets and such will only work after pressing enter. So I came up with this instead:

```
#!/usr/bin/php -q

<?php

function inKey($vals) {

    $inKey = "";

    While(!in_array($inKey,$vals)) {

        $inKey = trim(`read -s -n1 valu;echo \$valu`);

    }

    return $inKey;

}

function echoAT($Row,$Col,$prompt="") {

    // Display prompt at specific screen coords

    echo "\033[".$Row.";".$Col."H".$prompt;

}

    // Display prompt at position 10,10

    echoAT(10,10,"Opt : ");



    // Define acceptable responses

    $options = array("1","2","3","4","X");



    // Get user response

    $key = inKey($options);



    // Display user response & exit

    echoAT(12,10,"Pressed : $key\n");

?>
```

```
Hope this helps someone.
```

+ add a note