NAME

php - PHP Command Line Interface ’CLI’

php-cgi - PHP Common Gateway Interface ’CGI’ command

SYNOPSIS

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**

php [options] -S *addr:port* [-t *docroot*]

DESCRIPTION

PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. This is the command line interface that enables you to do the following:

You can parse and execute files by using parameter -f followed by the name of the *file* to be executed.

Using parameter -r you can directly execute PHP *code* simply as you would do inside a **.php** file when using the **eval()** function.

It is also possible to process the standard input line by line using either the parameter -R or -F. In this mode each separate input line causes the *code* specified by -R or the *file* specified by -F to be executed. You can access the input line by **$argn**. While processing the input lines **$argi** contains the number of the actual line being processed. Further more the parameters -B and -E can be used to execute *code* (see -r) before and after all input lines have been processed respectively. Notice that the input is read from **STDIN** and therefore reading from **STDIN** explicitly changes the next input line or skips input lines.

PHP also contains an built-in web server for application development purpose. By using the -S option where **addr:port** point to a local address and port PHP will listen to HTTP requests on that address and port and serve files from the current working directory or the **docroot** passed by the -t option.

If none of -r -f -B -R -F -E or -S is present but a single parameter is given then this parameter is taken as the filename to parse and execute (same as with -f). If no parameter is present then the standard input is read and executed.

OPTIONS

--interactive
-a Run PHP interactively. This lets you enter snippets of PHP code that directly get executed. When readline support is enabled you can edit the lines and also have history support.

--bindpath *address:port|port*
-b *address:port|port*
Bind Path for external FASTCGI Server mode (CGI only).

--no-chdir
-C Do not chdir to the script’s directory (CGI only).

--no-header
-q Quiet-mode. Suppress HTTP header output (CGI only).

--timing *count*
-T *count* Measure execution time of script repeated count times (CGI only).

--php-ini *path|file*
-c *path|file* Look for **php.ini** file in the directory *path* or use the specified *file*

--no-php-ini
-n No **php.ini** file will be used

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

-e Generate extended information for debugger/profiler

--file *file*
-f *file* Parse and execute *file*

--help
-h This help

--hide-args
-H Hide script name (*file*) and parameters (*args...*) from external tools. For example you may want to use this when a php script is started as a daemon and the command line contains sensitive data such as passwords.

--info
-i PHP information and configuration

--syntax-check
-l Syntax check only (lint)

--modules
-m Show compiled in modules

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

--process-begin *code*
-B *begin_code* Run PHP *begin_code* before processing input lines

--process-code *code*
-R *code* Run PHP *code* for every input line

--process-file *file*
-F *file* Parse and execute *file* for every input line

--process-end *code*
-E *end_code* Run PHP *end_code* after processing all input lines

--syntax-highlight
-s Output HTML syntax highlighted source

--server *addr:port*
-S *addr:port* Start built-in web server on the given local address and port

--docroot *docroot*
-t *docroot* Specify the document root to be used by the built-in web server

--version
-v Version number

--strip
-w Output source with stripped comments and whitespace

--zend-extension *file*
-z *file* Load Zend extension *file*

args... Arguments passed to script. Use **'--'** *args* when first argument starts with **'-'** or script is read from stdin

--rfunction *name*
--rf *name* Shows information about function **name**

--rclass *name*
--rc *name* Shows information about class **name**

--rextension *name*
--re *name* Shows information about extension **name**

--rzendextension
name
--rz *name* Shows information about Zend extension **name**

--rextinfo *name*
--ri *name* Shows configuration for extension **name**

--ini Show configuration file names

FILES

/etc/php/@PHP_MAJOR_VERSION@.@PHP_MINOR_VERSION@/cli/php.ini
The configuration file for the CLI version of PHP.

/etc/php/@PHP_MAJOR_VERSION@.@PHP_MINOR_VERSION@/cgi/php.ini
The configuration file for the CGI version of PHP.

/etc/php/@PHP_MAJOR_VERSION@.@PHP_MINOR_VERSION@/apache2/php.ini
The configuration file for the version of PHP that apache2 uses.

EXAMPLES

php -r 'echo "Hello World\n";'
This command simply writes the text "Hello World" to standard out.

php -r 'print_r(gd_info());'
This shows the configuration of your gd extension. You can use this to easily check which image formats you can use. If you have any dynamic modules you may want to use the same ini file that php uses when executed from your webserver. There are more extensions which have such a function. For dba use:
php -r 'print_r(dba_handlers(1));'

php -R 'echo strip_tags($argn)."\n";'

This PHP command strips off the HTML tags line by line and outputs the result. To see how it works you can first look at the following PHP command ´*php -d html_errors=1 -i*´ which uses PHP to output HTML formatted configuration information. If you then combine those two ´*php ...|php ...*´ you'll see what happens.

php -E 'echo "Lines: $argi\n";'

Using this PHP command you can count the lines being input.

php -R '@$l+=count(file($argn));' -E 'echo "Lines:$l\n";'

In this example PHP expects each input line being a file. It counts all lines of the files specified by each input line and shows the summarized result. You may combine this with tools like find and change the php scriptlet.

php -R 'echo "$argn\n"; fgets(STDIN);'

Since you have access to STDIN from within -B -R -F and -E you can skip certain input lines with your code. But note that in such cases $argi only counts the lines being processed by php itself. Having read this you will guess what the above program does: skipping every second input line.

TIPS

You can use a shebang line to automatically invoke php from scripts. Only the CLI version of PHP will ignore such a first line as shown below:

```
#!/bin/php
<?php
// your script
?>
```

SEE ALSO

For a more or less complete description of PHP look here:
http://www.php.net/manual/

BUGS

You can view the list of known bugs or report any new bug you found at:
http://bugs.php.net

AUTHORS

The PHP Group: Thies C. Arntzen, Stig Bakken, Andi Gutmans, Rasmus Lerdorf, Sam Ruby, Sascha Schumann, Zeev Suraski, Jim Winstead, Andrei Zmievski.

Additional work for the CLI sapi was done by Edin Kadribasic, Marcus Boerger and Johannes Schlueter.

A List of active developers can be found here:
http://www.php.net/credits.php

And last but not least PHP was developed with the help of a huge amount of contributors all around the world.

VERSION INFORMATION

This manpage describes **php**, version 7.0.33-0+deb9u8.

COPYRIGHT

Copyright © 1997-2017 The PHP Group