|  |  |
| --- | --- |
|  |  |
| [A picture containing diagram  Description automatically generated](https://www.raspberry-pi-geek.com/var/rpi/storage/images/archive/2014/07/php-on-raspberry-pi/figure-1/13007-1-eng-US/Figure-1_lightbox.png) | [Graphical user interface, text, application, chat or text message  Description automatically generated](https://www.raspberry-pi-geek.com/var/rpi/storage/images/archive/2014/07/php-on-raspberry-pi/figure-2/13010-1-eng-US/Figure-2_lightbox.png) |

Before you can start working with PHP on Raspberry Pi, you need to install PHP itself on the machine.

In most situations, PHP is used in tandem with a web server like Apache. Installing both PHP and Apache on Raspberry Pi is a matter of running:

sudo apt-get install apache2 php5

Instead of Apache, you can install a more lightweight server like lighttpd using the

sudo apt-get install lighttpd php5

Building Bridges

The easiest way to use PHP with Raspberry Pi is through the shell\_exec() function which lets you execute shell commands, so it can act as a sort of bridge between PHP and the Raspberry Pi. In the most simple case, shell\_exec() can call Python scripts that perform certain tasks and control GPIO pins.

Another approach is to deploy the Wiring Pi library [[1]](https://www.raspberry-pi-geek.com/Archive/2014/07/PHP-on-Raspberry-Pi#article_i1) for working with GPIO pins and then use the library with PHP via the shell\_exec() function. For this solution to work, you need to install Wiring Pi on Raspberry Pi first. The library is not available as a binary package, so you need to compile and install it from the source. Start with installing the Git software using:

sudo apt-get install git-core

Then, clone the Wiring Pi Git repository by running

git clone git://git.drogon.net/wiringPi

Switch to the resulting wiringPi directory and use the ./build command to compile and install Wiring Pi:

cd wiringPi

./build

To make sure Wiring Pi is installed and works properly,

1. Run the gpio -v command; it should return the current version of Wiring Pi along with the basic Raspberry Pi info.
2. Next, execute the gpio readall command to view a detailed GPIO layout diagram.

To put Wiring Pi to practical use, I'll build a super-simple PHP app for controlling an LED. Connect an LED with a resistor to GPIO pin 17 and GND as shown in [Figure 1](https://www.raspberry-pi-geek.com/Archive/2014/07/PHP-on-Raspberry-Pi#article_f1). Open the terminal on your Raspberry Pi (or connect to it via SSH) and switch to the /var/www directory.

Then, use the sudo nano gpio.php command to create the gpio.php file for editing. Place the code in [Listing 1](https://www.raspberry-pi-geek.com/Archive/2014/07/PHP-on-Raspberry-Pi#article_l1) (adapted from the Raspberry Pi Tutorials website [[2]](https://www.raspberry-pi-geek.com/Archive/2014/07/PHP-on-Raspberry-Pi#article_i2)) in the file.

**Listing 1**

**Simple PHP App to Control an LED ( gpio.php file)**

01 <html>

02 <head>

03 <meta name="viewport" content="width=device-width" />

04 <title>LED Control</title>

05 </head>

06 <body>

07 LED Control:

08 <form method="get" action="gpio.php">

09 <input type="submit" value="ON" name="on">

10 <input type="submit" value="OFF" name="off">

11 </form>

12 <?php

13 $setmode17 = shell\_exec("/usr/local/bin/gpio -g mode 17 out");

14 if(isset($\_GET['on'])){

15 $gpio\_on = shell\_exec("/usr/local/bin/gpio -g write 17 1");

16 echo "LED is on";

17 }

18 else if(isset($\_GET['off'])){

19 $gpio\_off = shell\_exec("/usr/local/bin/gpio -g write 17 0");

20 echo "LED is off";

21 }

22 ?>

23 </body>

24 </html>

The key element of the app is an HTML form containing ON and OFF buttons. When you press one of the buttons, its value is passed as a part of the URL (gpio.php?on=ON and gpio.php?off=OFF).

GPIO pin 17 is controlled by PHP code that uses the shell\_exec() function. The statement in line 13 sets the mode of the pin to out; the code block in lines 14-21 reads the current value from the URL and uses the shell\_exec() function to turn the pin on and off.

Now, point the browser to *http://127.0.0.1/gpio.php* (replace 127.0.0.1 with the actual IP address of the Raspberry Pi) and use the buttons to turn the LED connected to the Raspberry Pi on and off (Figure  2).

[Graphical user interface, text, application, chat or text message

Description automatically generated](https://www.raspberry-pi-geek.com/var/rpi/storage/images/archive/2014/07/php-on-raspberry-pi/figure-2/13010-1-eng-US/Figure-2_lightbox.png)

Figure 2: Behold the simple PHP app in all its bare-bones glory.

If the buttons don't work, most likely the web server doesn't have appropriate rights to execute shell commands. To fix this, run the sudo visudo command and add the following line to the sudoers file:

www-data ALL=NOPASSWD: ALL

Also, make sure that the /var/www directory belongs to the www-data user and group (use sudo chown -R www-data:www-data /var/www to set the correct owner).

Even this very simple PHP app can be put to several uses with a minimum of tweaking. For example, I use a simple transistor switch connected to Raspberry Pi to control my film SLR camera [[3]](https://www.raspberry-pi-geek.com/Archive/2014/07/PHP-on-Raspberry-Pi#article_i3). The original solution used a simple Python script to control the switch. To run the script, I had to do this via an SSH connection, which wasn't very practical in many situations. So, I modified the example PHP script for use with the transistor switch (Listing  2).

**Listing 2**

**PHP App to Control a Transistor Switch**

01 <html>

02 <head>

03 <meta name="viewport" content="width=device-width" />

04 <title>Trigger</title>

05 </head>

06 <body>

07 Trigger switch:

08 <form method="get" action="switch.php">

09 <input type="submit" value="Trigger" name="switch">

10 </form>

11 <?php

12 $setmode17 = shell\_exec("/usr/local/bin/gpio -g mode 17 out");

13 if(isset($\_GET['switch'])){

14 $gpio\_off = shell\_exec("/usr/local/bin/gpio -g write 17 1");

15 sleep (0.5);

16 $gpio\_on = shell\_exec("/usr/local/bin/gpio -g write 17 0");

17 echo "Done!";

18 }

19 ?>

20 </body>

21 </html>

All I had to do was remove one of the buttons and modify commands in the if condition to turn on the switch, wait for 0.5 seconds (so the camera registers the signal), and then turn off the switch. Now, I can use the PHP app to trigger my camera from any computer and mobile device.

Taking the gpio-php Route

Although using Wiring Pi via shell\_exec() calls offers an easy way to control GPIO pins in PHP scripts, it's not the only option at your disposal. The php-gpio project [[4]](https://www.raspberry-pi-geek.com/Archive/2014/07/PHP-on-Raspberry-Pi#article_i4), for example, provides a dedicated PHP library for accessing GPIO pin on Raspberry Pi. To get started with php-gpio, use the following commands to install the library and the accompanying files into the /home/pi directory:

wget http://getcomposer.org/composer.phar

php composer.phar create-project --stability='dev'ronanguilloux/php-gpio

php-gpio uses a handful of simple API calls to set GPIO pins and change their states from a PHP script. To enable this functionality, however, the script must contain the following statements that load the library and set up a GPIO pin:

require 'vendor/autoload.php';

use PhpGpio\Gpio;

$gpio = new GPIO();

$gpio->setup(17, "out");

In this example, the last statement sets GPIO pin 17 for output. Controlling the pins requires two more commands:

$gpio->output(17, 1)

$gpio->output(17, 0)

The first sets the pin's state to 1 (i.e., turns it on), and the second changes the state to 0 (i.e., turns the pin off). Finally, the $gpio->unexportAll() command resets all pins.

Using these commands, you can quickly whip up a simple PHP script that blinks the LED connected to GPIO pin 17 ([Listing 3](https://www.raspberry-pi-geek.com/Archive/2014/07/PHP-on-Raspberry-Pi#article_l3)).

**Listing 3**

**Simple PHP Script to Blink LED**

01 <?php

02 require 'vendor/autoload.php';

03 use PhpGpio\Gpio;

04 $gpio = new GPIO();

05 $gpio->setup(17, "out");

06 while (true){

07 $gpio->output(17, 1);

08 sleep(1);

09 $gpio->output(17, 0);

10 sleep(1);

11 }

To run the script, execute the sudo php blinking\_led\_script.php command. Although the php-gpio library makes it easy to control GPIO pins from PHP scripts, keep in mind that the scripts themselves must be run from the command line. In other words, you cannot use the described commands directly in PHP pages served by a web server.

The solution is simple: Use the shell\_exec() function to call the script from a PHP app. For example, I run a simple PHP-based photo gallery on my Raspberry Pi, and I have added the shell\_exec('sudo php path/to/php-gpio/blink\_led.php') statement that calls the blink\_led.php script. This way, when someone visits my gallery, the LED blinks for a couple of seconds.

The only problem is that the entire PHP app pauses while the called script runs. In the case of the blinking LED script, that might not be an issue, but it could be a serious problem if you call a script that takes longer to complete. Fortunately, an easy fix is available. You can run the script in the background and discard output by redirecting the script:

shell\_exec('sudo php path/to/php-gpio/\

blink\_led.php > /dev/null 2> /dev/null &');

Again, because all PHP scripts on Raspberry Pi must be run with root privileges, you need to add the www-data user to the sudoers file.

The php-gpio-web GitHub repository, for example, contains a simple web app [[5]](https://www.raspberry-pi-geek.com/Archive/2014/07/PHP-on-Raspberry-Pi#article_i5) that demonstrates how to create a web interface to control an LED, and the temperature-pi project [[6]](https://www.raspberry-pi-geek.com/Archive/2014/07/PHP-on-Raspberry-Pi#article_i6) shows how to read and log data from a temperature sensor.