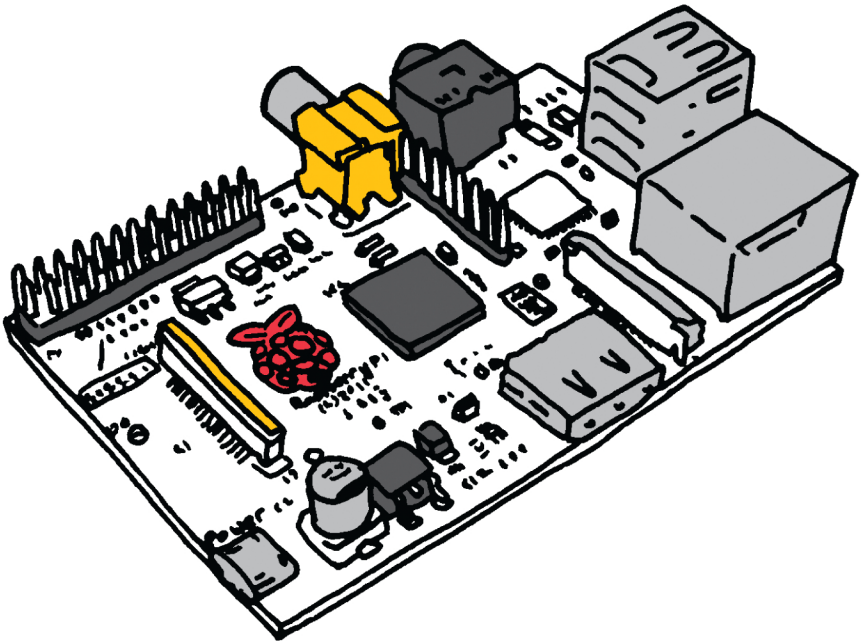
Make: PROJECTS

GETTING TO KNOW THE $35 ARM-POWERED LINUX COMPUTER

Getting Started with Raspberry Pi

Matt Richardson & Shawn Wallace



O'REILLY

Make: makezine.com

Hardware/Electronics

Make: PROJECTS

Getting Started with Raspberry Pi

What can you do with the Raspberry Pi, a $35 computer the size of a credit card? All sorts of things! If you're learning how to program, or looking to build new electronic projects, this hands-on guide will show you just how valuable this flexible little platform can be.

This book takes you step-by-step through many fun and educational possibilities. Take advantage of several preloaded programming languages. Use the Raspberry Pi with Arduino. Create Internet-connected projects. Play with multimedia. With Raspberry Pi, you can do all of this and more.

- Get acquainted with hardware features on the Pi's board
- Learn enough Linux to move around the operating system
- Pick up the basics of Python and Scratch—and start programming
- Draw graphics, play sounds, and handle mouse events with the Pygame framework
- Use the Pi's input and output pins to do some hardware hacking
- Discover how Arduino and the Raspberry Pi complement each other
- Integrate USB webcams and other peripherals into your projects
- Create your own Pi-based web server with Python

Matt Richardson is a creative technologist, video producer, and contributing editor to *MAKE* magazine and Makezine.com.

Shawn Wallace is an editor at O'Reilly Media.

Make: makezine.com

O'REILLY®

US $14.99 CAN $15.99

ISBN: 978-1-449-34421-4

7/Basic Input and Output

While the Raspberry Pi is, in essence, a very inexpensive Linux computer, there are a few things that distinguish it from laptop and desktop machines that we usually use for writing email, browsing the web, or word processing. One of the main differences is that the Raspberry Pi can be directly used in electronics projects because it has *general purpose input and output* pins right on the board, shown in Figure 7-1.

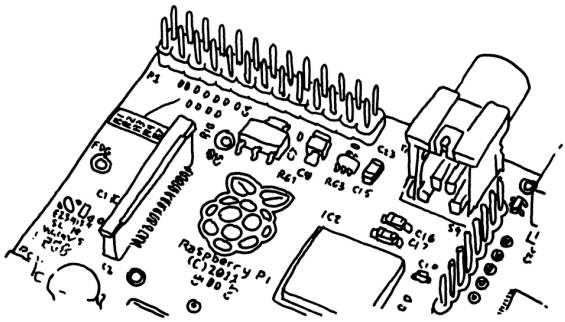


Figure 7-1. *Raspberry Pi's GPIO Pins*

These GPIO pins can be accessed for controlling hardware such as LEDs, motors, and relays, which are all examples of outputs. As for inputs, your Raspberry Pi can read the status of buttons, switches, and dials, or it can read sensors like temperature, light, motion, or proximity sensors (among many others).

One of the drawbacks to the Raspberry Pi is that there's no way to directly connect *analog sensors*, such as light and temperature sensors. Doing so requires a chip called an *analog to digital converter* or *ADC*. See Appendix C for how to read analog sensors using an ADC.

The best part of having a computer with GPIO pins is that you can create programs to read the inputs and control the outputs based on many different conditions, as easily as you'd program your desktop computer. Unlike a typical microcontroller board, which also has programmable GPIO pins, the Raspberry Pi has a few extra inputs and outputs such as your keyboard, mouse, and monitor, as well as the Ethernet port, which can act as both an input and an output. If you have experience creating electronics projects with microcontroller boards like the Arduino, you have a few more inputs and outputs at your disposal with the Raspberry Pi. Best of all, they're built right in; there's no need to wire up any extra circuitry to use them.

Having a keyboard, mouse, and monitor is not the only advantage that Raspberry Pi has over typical microcontroller boards. There are a few other key features that will help you in your electronics projects:

Filesystem
: Being able to read and write data in the Linux file system will make many projects much easier. For instance, you can connect a temperature sensor to the Raspberry Pi and have it take a reading once a second. Each reading can be appended to the end of a log file, which can be easily downloaded and parsed in a graphing program. It can even be graphed right on the Raspberry Pi itself!

Linux tools
: Packaged in the Raspberry Pi's Linux distribution is a set of core command-line utilities, which let you work with files, control processes, and automate many different tasks. These powerful tools are at your disposal for all of your projects. And since there is an enormous community of Linux users that depend on these core utilities, getting help is usually one web search away. For general Linux help, you can usually find answers at Stack Overflow. If you have a question specific to Raspberry Pi, try the Raspberry Pi Forum or the Raspberry Pi section of Stack Overflow.

Languages
: There are many programming languages out there and embedded Linux systems like the Raspberry Pi give you the flexibility to choose whichever language you're most comfortable with. The examples in this book will use shell scripting and Python but they could easily be translated to languages like C, Java, Perl, or many others.

Using Inputs and Outputs

There are a few supplies that you'll need in addition to the Raspberry Pi itself in order to try out these basic input and output tutorials. Many of these parts you'll be able to find in your local RadioShack, or they can be ordered online from stores like Maker Shed, Sparkfun, Adafruit, Mouser, or Digi-Key. Here are a few of the basic parts:

- Solderless breadboard
- LEDs, assorted
- Male-to-male jumper wires
- Female-to-male jumper wires (These are not as common as their male-to-male counterparts but are needed to connect the Raspberry Pi's GPIO pins to the breadboard.)
- Pushbutton switch
- Resistors, assorted

To make it easier to connect breadboarded components to the Raspberry Pi's pins, we also recommend Adafruit's Pi Cobbler Breakout Kit. This eliminates the need to use female-to-male jumper wires. The kit comes unassembled so it's up to you to solder the parts onto the board, but it's easy and Adafruit's tutorial walks you through the process step-by-step. The Pi Cobbler Breakout Kit is included, along with the other components listed above (with the exception of the female-to-male jumper wires, which are not needed if you have the Pi Cobbler Breakout Kit), in MAKE's Raspberry Pi Starter Kit (*http://oreil.ly/pikit*).

In Figure 7-2, we've labeled each pin according to its default GPIO signal number, which is how you'll refer to a particular pin in the commands you execute and in the code that you write. The unlabeled pins are assigned to other functions by default.

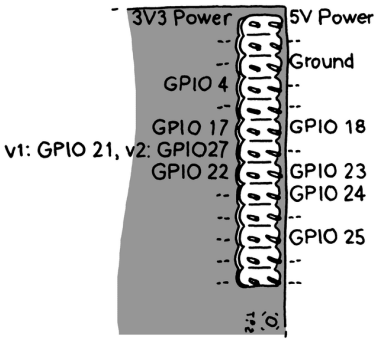


Figure 7-2. *The default GPIO pins on the Raspberry Pi. In recent revisions of the board, GPIO pin 21 was swapped for GPIO pin 27.*

You may have noticed that one of the pins has two different GPIO pin numbers in Figure 7-2. In recent versions of the board, GPIO pin 21 became GPIO pin 27. To determine the version of your board, type `cat /proc/cpuinfo` on the command line. If your revision number is listed as 0002 or 0003, you have the first version of the board. If you have a higher number, or a letter, you have a later version of the board.

Digital Output: Lighting Up an LED

The easiest way to use outputs with the GPIO pins is by connecting an LED, or light emitting diode. You can then use the Linux command line to turn the LED on and off. Once you have an understanding of how these commands work, you're one step closer to having an LED light up to indicate when you have new email, when you need to take an umbrella with you as you leave your house, or when it's time to go to bed. It's also very easy to go beyond a basic LED and use a relay to control a lamp on a set schedule, for instance.

Beginner's Guide to Breadboarding

If you've never used a breadboard (Figure 7-3) before, it's important to know which terminals are connected together. In the diagram below, we've shaded the terminal connections on a typical breadboard. Note that the power buses on the left side are not connected to the power buses on the right side of the breadboard. You'll have to use male-to-male jumper cables to connect them to each other if you need ground and voltage on both sides of the breadboard.

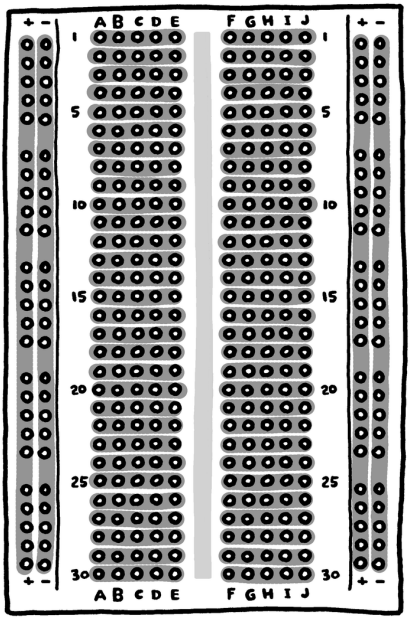


Figure 7-3. *Breadboard*

1. Using a male-to-female jumper wire, connect pin 25 on the Raspberry Pi to the breadboard. Refer to Figure 7-2 for the location of each pin on the Raspberry Pi's GPIO header.
2. Using another jumper wire, connect the Raspberry Pi's ground pin to the negative power bus on the breadboard.
3. Now you're ready to connect the LED (see Figure 7-4). Before you do that, it's important to know that LEDs are *polarized*: it matters which of the LED's wires is connected to what. Of the two leads coming off the LED, the longer one is the anode and should be connected to a GPIO pin. The shorter lead is the cathode and should be connected to ground. Another way to tell the difference is by looking from the top. The flat side of the LED indicates the cathode, the side that should be connected to ground. Insert the anode side of the LED into the breadboard in the same channel as the jumper wire from pin 25, which will connect pin 25 to the LED. Insert the cathode side of the LED into the ground power bus.

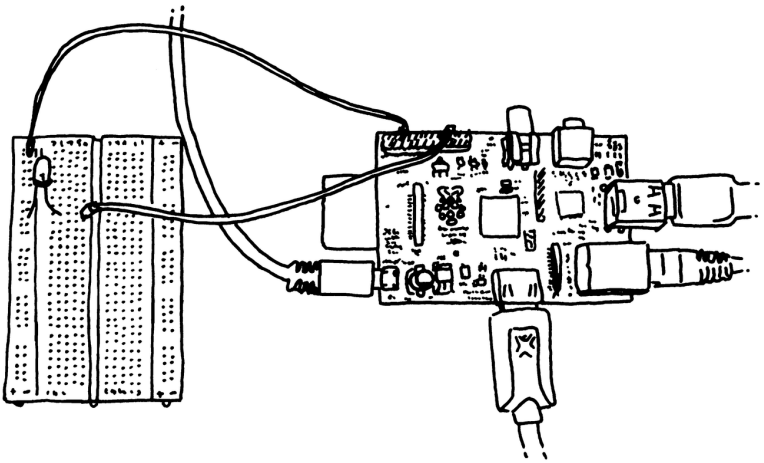


Figure 7-4. *Connecting an LED to the Raspberry Pi*

1. With your keyboard, mouse, and monitor hooked up, power on your Raspberry Pi and log in. If you're at a command line, you're ready to go. If you're in the X Window environment, double click on the LXTerminal icon on your desktop. This will bring up a terminal window.
2. In order to access the input and output pins from the command line, you'll need to run the commands as root, the *superuser* account on the Raspberry Pi. To start running commands as root, type **`sudo su`** at the command line and press enter:

   ```
   pi@raspberrypi ~ $ sudo su
   root@raspberrypi:/home/pi#
   ```

   You'll notice that the command prompt has changed, indicating that you're now running commands as root.

The root account has administrative access to all the functions and files on the system and there is very little protecting you from damaging the operating system if you type a command that can harm it, so exercise caution when running commands as root. If you do mess something up, don't worry about it too much; you can always reimage the SD card with a clean Linux install. When you're done working within the root account, type `exit` to return to working within the `pi` user account.

6. Before you can use the command line to turn the LED on pin 25 on and off, you need to *export the pin to the userspace* (in other words, make the pin available for use outside of the confines of the Linux kernel), this way:

```
root@raspberrypi:/home/pi# echo 25 > /sys/class/gpio/export
```

   The `echo` command writes the number of the pin you want to use (25) to the export file, which is located in the folder `/sys/class/gpio`. When you write pin numbers to this special file, it creates a new directory in `/sys/class/gpio` that has the control files for the pin. In this case, it created a new directory called `/sys/class/gpio/gpio25`.

7. Change to that directory with the `cd` command and list the contents of it with `ls`:

```
root@raspberrypi:/home/pi# cd /sys/class/gpio/gpio25
root@raspberrypi:/sys/class/gpio/gpio25# ls
active_low  direction  edge  power  subsystem  uevent  value
```

   The command `cd` stands for "change directory." It changes the working directory so that you don't have to type the full path for every file. `ls` will list the files and folders within that directory There are two files that you're going to work with in this directory: `direction` and `value`.

8. The `direction` file is how you'll set this pin to be an input (like a button) or an output (like an LED). Since you have an LED connected to pin 25 and you want to control it, you're going to set this pin as an output:

```
root@raspberrypi:/sys/class/gpio/gpio25# echo out > direction
```

9. To turn the LED on, you'll use the `echo` command again to write the number 1 to the `value` file:

```
root@raspberrypi:/sys/class/gpio/gpio25# echo 1 > value
```

10. After pressing enter, the LED will turn on! Turning it off is as simple as using `echo` to write a zero to the value file:

```
root@raspberrypi:/sys/class/gpio/gpio25# echo 0 > value
```

Virtual Files

The files that you're working with aren't actually files on the Raspberry Pi's SD card, but rather are a part of Linux's *virtual file system*, which is a system that makes it easier to access low-level functions of the board in a simpler way. For example, you could turn the LED on and off by writing to a particular section of the Raspberry Pi's memory, but doing so would require more coding and more caution.

So if writing to a file is how you control components that are outputs, how do you check the status of components that are inputs? If you guessed "reading a file," then you're absolutely right. Let's try that now.

Digital Input: Reading a Button

Simple pushbutton switches like the one in Figure 7-5 are great for controlling basic digital input and best of all, they're made to fit perfectly into a breadboard.

These small buttons are very commonly used in electronics projects and understanding what's going on inside of them will help you as you prototype your project. When looking at the button as it sits in the breadboard (see Figure 7-5): the top two terminals are always connected to each other. The same is true for the bottom two terminals; they're always connected. When you push down on the button, these two sets of terminals are connected to each other.

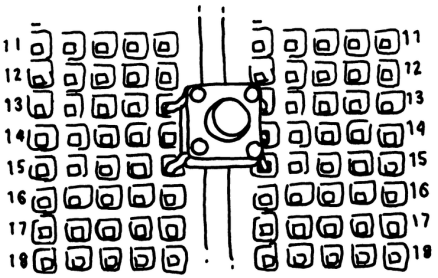


Figure 7-5. *Button*

When you read a digital input on a Raspberry Pi, you're checking to see if the pin is connected to either 3.3 volts or to ground. It's important to remember that it must be either one or the other, and if you try to read a pin that's not connected to either 3.3 volts or ground, you'll get unexpected results. Once you understand how digital input with a pushbutton works, you can start using components like magnetic security switches, arcade joysticks, or even vending machine coin accepters.

1. Insert the pushbutton into the breadboard so that its leads straddle the middle channel.
2. Using a jumper wire, connect pin 24 from the Raspberry Pi to one of the top terminals of the button.
3. Connect the 3V3 pin from the Raspberry Pi to the positive power bus on the breadboard.

Be sure that you connect the button to the 3V3 pin and not the 5V pin. Using more than 3.3 volts on an input pin will permanently damage your Raspberry Pi.

4. Connect one of the bottom terminals of the button to the power bus. Now when you push down on the button, the 3.3 volts will be connected to pin 24.
5. Remember what we said about how a digital input must be connected to *either* 3.3 volts or ground? When you let go of the button, pin 24 isn't connected to either of those and is therefore *floating*. This condition will cause unexpected results, so let's fix that. Use a 10K resistor (labeled with the colored bands: brown, black, orange, and then silver or gold) to connect the input side of the switch to the ground rail, which you connected to the Raspberry Pi's ground in the output example. When the switch is not pressed, the pin will be connected to ground.

   Since electricity always follows the path of least resistance towards ground, when you press the switch, the 3.3 volts will go towards the Raspberry Pi's input pin, which has less resistance than the 10K resistor. When everything's hooked up, it should look like Figure 7-6.

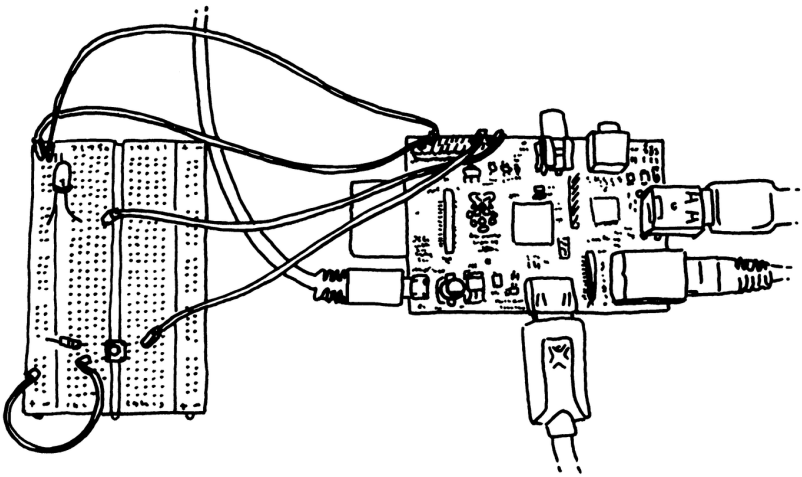


Figure 7-6. *Connecting a button to the Raspberry Pi*

6. Now that the circuit is built, let's read the value of the pin from the command line. If you're not already running commands as root, type **sudo su**.
7. As with the previous example, you need to export the input pin to user-space:

```
root@raspberrypi:/sys/class/gpio/gpio25# echo 24 > /sys/class/gpio/
export
```

8. Let's change to the directory that was created during the export operation:

```
root@raspberrypi:/sys/class/gpio/gpio25# cd /sys/class/gpio/gpio24
```

9. Now set the direction of the pin to input:

```
root@raspberrypi:/sys/class/gpio/gpio24# echo in > direction
```

10. To read the value of the of the pin, you'll use the `cat` command, which will print the contents of files to the terminal. The command `cat` gets its name because it can also be used to concatenate, or join, files together. It can also display the contents of a file for you.

```
root@raspberrypi:/sys/class/gpio/gpio24# cat value
0
```

11. The zero indicates that the pin is connected to ground. Now press and hold the button while you execute the command again:

```
root@raspberrypi:/sys/class/gpio/gpio24# cat value
1
```

12. If you see the number one, you'll know you've got it right!

To easily execute a command that you've previously executed, hit the up key until you see the command that you want to execute and hit enter.

Now that you can use the Linux command line to control an LED or read the status of a button, let's use a few of Linux's built-in tools to create a very simple project that uses digital input and output.

Project: Cron Lamp Timer

Let's say you're leaving for a long vacation early tomorrow morning and you want to ward off would-be burglars from your home. A lamp timer is a good deterrent, but hardware stores are closed for the night and you won't have time to get one before your flight in the morning. However, since you're a Raspberry Pi hobbyist, you have a few supplies lying around, namely:

- Raspberry Pi board
- Breadboard
- Jumper wires, female-to-male.
- PowerSwitch Tail II relay
- Hookup wire

With these supplies, you can make your own programmable lamp timer using two powerful Linux tools: *shell scripts* and *cron*.

Scripting Commands

A shell script is a file that contains a series of commands (just like the ones you've been using to control and read the pins). Take a look at the shell script below and the explanation of the key lines.

```
#!/bin/bash # ❶
echo Exporting pin $1. # ❷
echo $1 > /sys/class/gpio/export # ❸
echo Setting direction to out.
echo out > /sys/class/gpio/gpio$1/direction # ❹
echo Setting pin high.
echo 1 > /sys/class/gpio/gpio$1/value
```

❶ This line is required for all shell scripts.

❷ "$1" refers to the first command line argument.

❸ Instead of exporting a specific pin number, the script uses the first command line argument.

❹ Notice that the first command line argument replaces the pin number here as well.

Save that as a text file called **on.sh** and make it executable with the chmod command:

```
root@raspberrypi:/home/pi# chmod +x on.sh
```

You still need to be executing theses commands as root. Type **sudo su** if you're get errors like "Permission denied."

A command line argument is a way of passing information into a program or script by typing it in after name of the command. When you're writing a shell script, $1 refers to the first command line argument, $2 refers to the second, and so on. In the case of **on.sh**, you'll type in the pin number that you want to export and turn on. Instead of *hard coding* pin 25 into the shell script, it's more universal by referring to the pin that was typed in at the command line. To export pin 25 and turn it on, you can now type:

```
root@raspberrypi:/home/pi# ./on.sh 25 ❶
Exporting pin 25.
Setting direction to out.
Setting pin high.
```

❶ The "./" before the filename indicates that you're executing the script in the directory you're in.

If you still have the LED connected to pin 25 from earlier in the chapter, it should turn on. Let's make another shell script called `off.sh`, which will turn the LED off. It will look like this:

```
#!/bin/bash
echo Setting pin low.
echo 0 > /sys/class/gpio/gpio$1/value
echo Unexporting pin $1
echo $1 > /sys/class/gpio/unexport
```

Now let's make it executable and run the script:

```
root@raspberrypi:/home/pi/temp# chmod +x off.sh
root@raspberrypi:/home/pi/temp# ./off.sh 25
Setting pin low.
Unexporting pin 25
```

If everything worked, the LED should have turned off.

Connecting a Lamp

Of course, a tiny little LED isn't going to give off enough light to fool burglars into thinking that you're home, so let's hook up a lamp to the Raspberry Pi.

1. Remove the LED connected to pin 25.
2. Connect two strands of hookup wire to the breadboard, one that connects to pin 25 of the Raspberry Pi and the other to the ground bus.
3. The strand of wire that connects to pin 25 should be connected to the "+in" terminal of the PowerSwitch Tail.
4. The strand of wire that connects to ground should be connected to the "-in" terminal of the PowerSwitch Tail. Compare your circuit to Figure 7-7.
5. Plug the PowerSwitch Tail into the wall and plug a lamp into the PowerSwitch Tail. Be sure the lamp's switch is in the on position.
6. Now when you execute `./on.sh 25`, the lamp should turn on and if you execute `./off.sh 25`, the lamp should turn off!

Inside the PowerSwitch Tail there are a few electronic components that help you control high voltage devices like a lamp or blender by using a low voltage signal such as the one from the Raspberry Pi. The "click" you hear from the PowerSwitch Tail when it's turned on or off is the relay, the core component of the circuit inside. A relay acts like a switch for the high voltage device that can be turned on or off depending on whether the low voltage control signal from the Raspberry Pi is on or off.

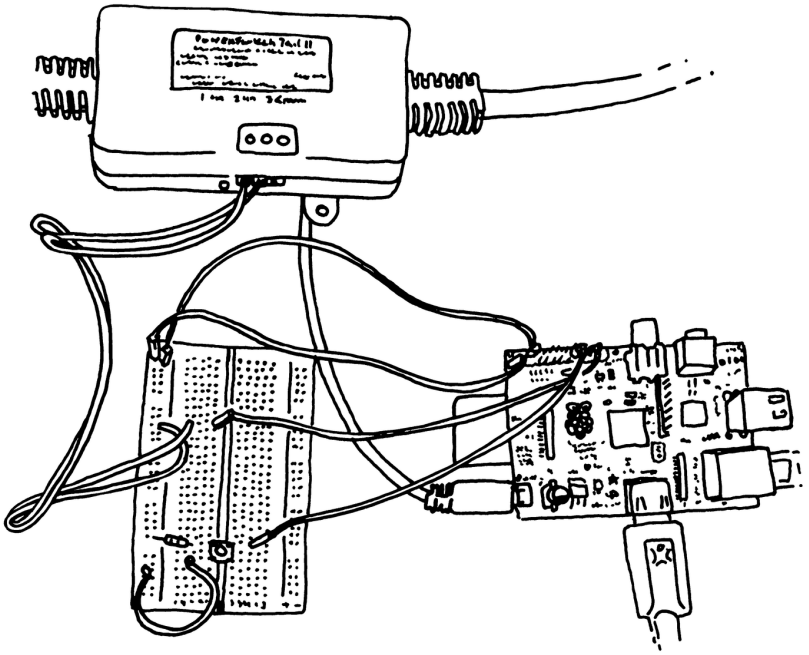


Figure 7-7. *Connecting a PowerSwitch Tail II to the Raspberry Pi*

Scheduling Commands with cron

So now you've packaged up a few different commands into two simple commands that can turn a pin on or off. And with the lamp connected to the Raspberry Pi through the PowerSwitch Tail, you can turn the lamp on or off with a single command. Now you can use `cron` to schedule the light to turn on and off at different times of day. `cron` is Linux's job scheduler. With it, you can set commands to execute on specific times and dates, or you can have jobs run on a particular period (for example, once an hour). You're going to schedule two jobs; one of them will turn the light on at 8:00pm and the other will turn the light off at 2:00 am.

As with other time-dependent programs, you'll want to make sure you've got the correct date and time set up on your Raspberry Pi, as described in Chapter 1.

To add these jobs, you'll have to edit the cron table (a list of commands that Linux executes at specified times):

```
root@raspberrypi:/home/pi/# crontab -e
```

This will launch a text editor to change root's cron table. At the top of the file, you'll see some information about how to modify the cron table. Use your arrow keys to get to the bottom of the file and add these two entries at the end of the file.

```
0 20 * * * /home/pi/on.sh 25
0 2 * * * /home/pi/off.sh 25
```

`cron` will ignore any lines that start with the hash mark. If you want to temporarily disable a line without deleting it or add a comment to the file, put a hash mark in front of the line.

Type `Control-X` to exit, type `y` to save the file when it prompts you, and hit enter to accept the default file name. When the file is saved and you're back at the command line, it should say `installing new crontab` to indicate that the changes you've made are going to be executed by `cron`.

More About Cron

Cron will let you schedule jobs for specific dates and times or on intervals. There are five time fields (or six if you want to schedule by year), each separated by a space followed by another space then the command to execute. Asterisks indicate that the job should execute each period. For example:

Table 7-1. *Cron Entry for Turning Light On at 8:00pm Every Day*

0	**20**	*****	*****	*****	**/home/pi/on.sh 25**
Minute (:00)	Hour (8pm)	Every Day	Every Month	Every Day of Week	path to command

Let's say you only wanted the lamp to turn on every weekday. Here's what the crontab entry would look like:

Table 7-2. *Cron Entry for Turning Light On at 8:00pm Every Weekday*

0	**20**	*****	*****	**1-5**	**/home/pi/on.sh 25**
Minute (:00)	Hour (8pm)	Every Day	Every Month	Monday to Friday	path to command

Let's say you have a shell script that checks if you have new mail and emails you if you do. Here's how you'd get that script to run every five minutes:

Table 7-3. *Cron Entry for Checking for Mail Every Five Minutes*

***/5**	*****	*****	*****	*****	**/home/pi/ checkMail.sh**
Every five minutes	Every Hour	Every Day	Every Month	Every Day of Week	path to command

The */5 indicates a period of every five minutes.

As you can see, `cron` is a powerful tool that's at your disposal for scheduling jobs for specific dates or times and scheduling jobs to happen on a specific interval.

Going Further

eLinux's Raspberry Pi GPIO Reference Page
: This is the most comprehensive reference guide to the Raspberry Pi's GPIO pins.

Adafruit: MCP230xx GPIO Expander on the Raspberry Pi
: If you don't have enough pins to work with, Adafruit offers this guide to using the MCP23008 chip for 8 extra GPIO pins and the MCP23017 for 16 extra GPIO pins.

About the Authors

Matt Richardson is a Brooklyn-based creative technologist and video producer. He's a contributor to *MAKE* magazine and Makezine.com. Matt is also the owner of Awesome Button Studios, a technology consultancy. Highlights from his work include the Descriptive Camera, a camera which outputs a text description of a scene instead of a photo. He also created The Enough AIready, a DIY celebrity-silencing device. Matt's work has garnered attention from *The New York Times*, *Wired*, *New York Magazine* and has also been featured at The Nevada Museum of Art and at the Santorini Bienniele. He is currently a Master's candidate at New York University's Interactive Telecommunications Program.

Shawn Wallace is an editor at O'Reilly and lives in Providence, RI. He is also a member of the Fluxama artist collective responsible for new iOS musical instruments such as Noisemusick and Doctor Om. He designed open hardware kits at Modern Device and taught the Fab Academy at the Providence Fab Lab. For years he was the managing director of the AS220 art space and is a cofounder of the SMT Computing Society.

The cover and body font is BentonSans, the heading font is Serifa, and the code font is Bitstreams Vera Sans Mono.