Raspberry Pi OS

Introduction

Edit this on GitHub

Raspberry Pi OS is a free, Debian-based operating system optimised for the Raspberry Pi hardware. Raspberry Pi OS supports over 35,000 Debian packages. We recommend Raspberry Pi OS for most Raspberry Pi use cases.

Because Raspberry Pi OS is derived from Debian, it follows a staggered version of the Debian release cycle. Releases happen roughly every 2 years.

The latest version of Raspberry Pi OS is based on Debian Bookworm. The previous version was based on Debian Bullseye.

You can find images of Raspberry Pi OS at raspberrypi.com/software/operating-systems/

Update software

Edit this on GitHub

Always keep the software running on your Raspberry Pi updated to the latest version. This keeps your device secure from vulnerabilities and ensures that you get the latest bug fixes.

Manage software packages with APT

Advanced Package Tool (APT) is the recommended way to install, update, and remove software in Raspberry Pi OS. You can access APT through the `apt` CLI.

Install updates

`apt` stores a list of software sources in a file at `/etc/apt/sources.list`. Before installing software, run the following command to **update** your local list of packages using `/etc/apt/sources.list`:

```
$ sudo apt update
```

Run the following command to **upgrade** all your installed packages to their latest versions:

```
$ sudo apt full-upgrade
```

TIP

Unlike Debian, Raspberry Pi OS is under continual development. As a result, package dependencies sometimes change, so you should always use `full-upgrade` instead of the standard `upgrade`.

Run these commands regularly to keep your software up-to-date. Using `apt` to keep Raspberry Pi OS up to date also keeps your Linux kernel and firmware up to date, since Raspberry Pi distributes them as Debian packages.

When Raspberry Pi releases a new major version of Raspberry Pi OS, the above commands won't upgrade your operating system to that new major version. To upgrade to a new major version, follow our OS upgrade instructions.

Search for software

To search the archives for a package, pass a search keyword to `apt-cache search`:

```
$ apt-cache search <keyword>
```

For example, consider the following search for the keyword "raspi":

```
$ apt-cache search raspi
raspi3-firmware - Raspberry Pi 2 and 3 GPU firmware and bootloaders
libcamera-apps - libcamera-apps
libcamera-apps-lite - libcamera-apps-lite
python-picamera - Pure Python interface to the Raspberry Pi's camera module.
python-picamera-docs - Documentation for the Python interface to the RPi's came
ra module.
python3-picamera - Pure Python interface to the Raspberry Pi's camera module.
raspi-config - Raspberry Pi configuration tool
raspi-gpio - Dump the state of the BCM270x GPIOs
raspi-gpio-dbgsym - debug symbols for raspi-gpio
raspinfo - Dump information about the Pi
rc-gui - raspi-config GUI
raspi-copies-and-fills - ARM-accelerated versions of selected functions from st
ring.h
raspi-copies-and-fills-dbgsym - debug symbols for raspi-copies-and-fills
```

The search returned multiple packages with names or descriptions that included the keyword.

Use the following command to view detailed information about a package:

```
$ apt-cache show <package-name>
```

For example, consider the following query for the "raspi-config" package:

```
$ apt-cache show raspi-config
Package: raspi-config
Version: 20210212
Architecture: all
Maintainer: Serge Schneider <serge@raspberrypi.org>
Installed-Size: 121
Depends: whiptail, parted, lua5.1, alsa-utils, psmisc, initramfs-tools
Recommends: triggerhappy, iw
Priority: optional
Section: utils
Filename: pool/main/r/raspi-config/raspi-config_20210212_all.deb
Size: 27976
SHA256: 772d4fd3c6d8c9da47ac56012b74e7828b53c8521ff1c47266bb38ec71750c10
SHA1: 08254c976a8260bde914c2df72f92ffb9317fef6
MD5sum: 80aaac13be6a9b455c822edb91cf8ea2
Description: Raspberry Pi configuration tool
```

```
 A simple configuration tool for common Raspberry Pi administrative tasks
Description-md5: 19630c04463bfe7193152448b53d85a0
```

Use this command to verify that the maintainer, version, and size match your expectations for a package.

Install a package

To install a package on your Raspberry Pi, pass the name of the package to the following command:

```
$ sudo apt install <package-name>
```

`apt` will display the amount of disk space the package will consume. Enter **Y** and press **Enter** to confirm installation of the package. You can skip this confirmation step by adding the `-y` flag to the command above.

Uninstall a package

To uninstall a package from your Raspberry Pi, pass the name of the package to the following command:

```
$ sudo apt remove <package-name>
```

TIP

> To completely remove all traces of the package, including configuration files, use `purge` instead of `remove`.

`apt` will display the amount of disk space removing the package will free up. Enter **Y** and press **Enter** to confirm removal of the package. You can skip this confirmation step by adding the `-y` flag to the command above.

Manage `apt` disk usage

Before running, `sudo apt full-upgrade` shows the amount of data you'll need to download and store on disk to complete an upgrade. To check that you have enough free disk space, run the following command:

```
$ df -h
```

`apt` stores downloaded package (`.deb`) files in `/var/cache/apt/archives`. During installation, `apt` downloads these packages, then copies files from the packages to the correct installation locations. Depending on the software you have installed, package files can take up significant amounts of space. To delete any lingering package files, run the following command:

```
$ sudo apt clean
```

Upgrade your operating system to a new major version

WARNING

> Before attempting a major version upgrade, make a backup.

To update the operating system to a new major release on your Raspberry Pi, image a second SD card with the new release. Use a USB SD card reader or network storage to copy files and configuration from your current installation to the new SD card. Then, swap the new SD card into the slot on your Raspberry Pi, and boot.

Upgrade your firmware

WARNING

> Before attempting a firmware upgrade, make a backup.

WARNING

> Pre-release versions of software are not guaranteed to work. Do not use `rpi-update` on any system unless recommended to do so by a Raspberry Pi engineer. It could leave your system unreliable or broken. Do not use `rpi-update` as part of any regular update process.

To update the firmware on your Raspberry Pi to the latest version, use `rpi-update`.

`rpi-update` downloads the latest pre-release version of the Linux kernel, its matching modules, device tree files, and the latest versions of the VideoCore firmware. It then installs these files into an existing Raspberry Pi OS install.

All the source data used by `rpi-update` comes from the `rpi-firmware` repository. This repository contains a subset of the data from the official firmware repository.

Run `rpi-update` as root to initiate the update. Once the update is complete, reboot your Raspberry Pi for these changes to take effect:

```
$ sudo rpi-update
$ sudo reboot
```

WHITE PAPER

Updating Raspberry Pi firmware

This whitepaper documents how to update the VideoCore firmware in a Raspberry Pi OS image.

Downgrade firmware to the last stable release

If you update your firmware to the latest release and experience an issue, use the following command to return to the last stable firmware release:

```
$ sudo apt update
$ sudo apt install --reinstall raspi-firmware
```

NOTE

If you still run Raspberry Pi OS Bullseye, you must instead reinstall `raspberrypi-kernel` using the following command:

```
$ sudo apt install --reinstall libraspberrypi0 libraspberrypi-{bin,dev,doc} raspberrypi-{kernel,bootloader}
```

Reboot your Raspberry Pi with `sudo reboot` to put these changes into effect.

Play audio and video

Edit this on GitHub

Raspberry Pi OS comes with VLC media player pre-installed. You can use VLC to play video and audio files. VLC uses hardware acceleration in Raspberry Pi OS, and supports many popular audio and video file formats.

VLC media player

VLC GUI

To play an audio or video file from Raspberry Pi Desktop, double-click on a file in the file manager. This automatically launches VLC to play the file. Alternatively, from the **Sound & Video** menu, launch **VLC Media Player**. Then, from the **Media** menu, select **Open File…** and navigate to the file you want to play.

By default, Raspberry Pi OS sends audio to your monitor over HDMI. To output audio to a different interface, such as the headphone jack or USB speakers, right-click on the speaker icon in the system tray and select an option.

`vlc` CLI

You can also launch VLC from the command line. For the examples below, we used a short clip from Big Buck Bunny. To download this clip from Raspberry Pi, run the following command:

```
$ wget --trust-server-names http://rptl.io/big-buck-bunny
```

To play the clip in VLC from the command line, run the following command:

```
$ vlc big-buck-bunny-1080p.mp4
```

To prevent the VLC GUI staying open after your file has finished playing, add the `--play-and-exit` flag:

```
$ vlc --play-and-exit big-buck-bunny-1080p.mp4
```

To play a video in fullscreen mode (which can result in smoother playback in some circumstances), add the `--fullscreen` flag:

```
$ vlc --play-and-exit --fullscreen big-buck-bunny-1080p.mp4
```

Use `cvlc` to play media without a GUI

If you use `cvlc` instead of `vlc` with any of these commands, then the VLC GUI won't be shown:

```
$ cvlc --play-and-exit big-buck-bunny-1080p.mp4
```

Play audio and video on Raspberry Pi OS Lite

Unlike the full version of Raspberry Pi OS, VLC doesn't come pre-installed on Raspberry Pi OS Lite. To play video and audio on Raspberry Pi OS Lite with VLC, install the required packages for playback without a desktop:

```
$ sudo apt install --no-install-recommends vlc-bin vlc-plugin-base
```

For the examples below, we used a short audio clip. To download this clip from Raspberry Pi, run the following command:

```
$ wget --trust-server-names http://rptl.io/startup-music
```

To play the clip in VLC from the command line, run the following command:

```
$ cvlc --play-and-exit computer-startup-music.mp3
```

Specify an audio output device

To force audio output to a particular device, pass the `alsa` value to the the `-A` option to use ALSA audio output, and the `--alsa-audio-device` option to specify an audio output device:

```
$ cvlc --play-and-exit -A alsa --alsa-audio-device <alsa-device> computer-start
up-music.mp3
```

Replace the `<alsa-device>` placeholder with one of the following options:

ALSA device	Description
`sysdefault:CARD=Headphones`	The headphone jack

ALSA device	Description
`sysdefault:CARD=vc4hdmi`	The HDMI output on a Raspberry Pi Zero, or Raspberry Pi Model 1, 2 or 3
`sysdefault:CARD=vc4hdmi0`	The HDMI0 output on a Raspberry Pi 4, 5, 400, or Compute Module 4
`sysdefault:CARD=vc4hdmi1`	The HDMI1 output on a Raspberry Pi 4, 5, 400, or Compute Module 4

TIP

Use the following command to get a list of all ALSA devices on your Raspberry Pi:

```
$ aplay -L | grep sysdefault
```

Specify a video output device

To force the video output to a particular device, use the `--drm-vout-display` option to specify a video output device:

```
$ cvlc --play-and-exit --drm-vout-display <drm-device> big-buck-bunny-1080p.mp4
```

Replace the `<drm-device>` placeholder with one of the following options:

DRM device	Description
`HDMI-A-1`	The HDMI output on a Raspberry Pi Zero, or Raspberry Pi Model 1, 2 or 3; **or** the HDMI0 output on a Raspberry Pi 4, 5, or 400
`HDMI-A-2`	The HDMI1 output on a Raspberry Pi 4, 5, or 400
`DSI-1`	The Raspberry Pi Touch Display

TIP

Use the following command to get a list of all DRM devices on your Raspberry Pi:

```
$ kmsprint | grep Connector
```

Specify both audio and video output devices

You can combine audio and video output options. For example, to direct video output to the touchscreen, and audio output to the headphone jack, use the following combination of the commands above:

```
$ cvlc --play-and-exit --fullscreen --drm-vout-display DSI-1 -A alsa --alsa-aud
io-device sysdefault:CARD=Headphones your_video.mp4
```

Improve stream playback performance

If you have a raw H.264 stream, like those captured from a Raspberry Pi Camera Module, you can improve playback performance in VLC by wrapping the stream inside a container format such as MP4. You can use `ffmpeg` to convert stream content into a container file. For example, the following command converts a stream named `video.h264` to a MP4 container named `video.mp4` at 30fps:

```
$ ffmpeg -r 30 -i video.h264 -c:v copy video.mp4
```

Utilities

Edit this on GitHub

There are several useful command-line utilities pre-installed in Raspberry Pi OS.

kmsprint

The `kmsprint` tool can be used to list the display-modes supported by the monitors attached to the Raspberry Pi. Use `kmsprint` to see details of the monitors connected to the Raspberry Pi, and `kmsprint -m` to see a list of all the display modes supported by each monitor. You can find source code for the `kmsprint` utility on Github.

vclog

`vclog` displays log messages from the VideoCore GPU from Linux running on the Arm. It needs to be run as root.

`sudo vclog --msg` prints out the message log, whilst `sudo vclog --assert` prints out the assertion log.

vcgencmd

The `vcgencmd` tool is used to output information from the VideoCore GPU on the Raspberry Pi. You can find source code for the `vcgencmd` utility on GitHub.

To get a list of all commands supported by `vcgencmd`, use `vcgencmd commands`. Some useful commands and their required parameters are listed below.

vcos

The `vcos` command has two useful sub-commands:

- `version` displays the build date and version of the firmware on the VideoCore
- `log status` displays the error log status of the various VideoCore firmware areas

`version`

Displays the build date and version of the VideoCore firmware.

`get_throttled`

Returns the throttled state of the system. This is a bit pattern. A bit being set indicates the following meanings:

Bit	Hexadecimal value	Meaning
0	`0x1`	Undervoltage detected
1	`0x2`	Arm frequency capped
2	`0x4`	Currently throttled
3	`0x8`	Soft temperature limit active
16	`0x10000`	Undervoltage has occurred
17	`0x20000`	Arm frequency capping has occurred
18	`0x40000`	Throttling has occurred
19	`0x80000`	Soft temperature limit has occurred

`measure_temp`

Returns the temperature of the SoC as measured by its internal temperature sensor. On Raspberry Pi 4, `measure_temp pmic` returns the temperature of the PMIC.

`measure_clock [clock]`

This returns the current frequency of the specified clock. Accepts the following clock values:

clock	Description
`arm`	ARM core(s)
`core`	GPU core
`h264`	H.264 block
`isp`	Image Sensor Pipeline
`v3d`	3D block
`uart`	UART
`pwm`	PWM block (analogue audio output)
`emmc`	SD card interface
`pixel`	Pixel valves

clock	Description
`vec`	Analogue video encoder
`hdmi`	HDMI
`dpi`	Display Parallel Interface

e.g. `vcgencmd measure_clock arm`

`measure_volts [block]`

Displays the current voltages used by the specific block. Accepts the following block values:

block	Description
`core`	VC4 core voltage
`sdram_c`	SDRAM Core Voltage
`sdram_i`	SDRAM I/O voltage
`sdram_p`	SDRAM Phy Voltage

`otp_dump`

Displays the content of the OTP (one-time programmable) memory inside the SoC. These are 32-bit values, indexed from 8 to 64. See the OTP bits page for more details.

`get_config [configuration item|int|str]`

Displays the value of the configuration setting specified: alternatively, specify either `int` (integer) or `str` (string) to see all configuration items of the given type. For example, the following command returns the total memory on the device in megabytes:

```
$ vcgencmd get_config total_mem
```

`get_mem type`

Reports on the amount of memory addressable by the Arm and the GPU. To show the amount of Arm-addressable memory, use `vcgencmd get_mem arm`; to show the amount of GPU-addressable memory, use `vcgencmd get_mem gpu`. On devices with more than 1GB of memory, the `arm` parameter will always return 1GB minus the `gpu` memory value, since the GPU firmware is only aware of the first 1GB of memory. To get an accurate report of the total memory on the device, see the `total_mem` configuration item and the `get_config` section above.

`codec_enabled [type]`

Reports whether the specified codec type is enabled. Possible options for type are AGIF, FLAC, H263, H264, MJPA, MJPB, MJPG, MPG2, MPG4, MVC0, PCM, THRA, VORB, VP6,

VP8, WMV9, WVC1. Note that because the H.265 HW block on the Raspberry Pi 4 and 400 is not part of the VideoCore GPU, its status is not accessed via this command.

`mem_oom`

Displays statistics on any OOM (out of memory) events occurring in the VideoCore memory space.

`mem_reloc_stats`

Displays statistics from the relocatable memory allocator on the VideoCore.

`read_ring_osc`

Returns the current speed, voltage and temperature of the ring oscillator.

Accessibility options

Edit this on GitHub

Visual aids

Users of Raspberry Pi OS with visual impairments can find helpful tools in the **Recommended Software** menu.

We offer Orca screen reader to ease navigation of Raspberry Pi Desktop. Additionally, we offer screen magnifier to increase the readability of UI and screen elements.

Orca screen reader

You can install Orca screen reader from the **Recommended Software** section of the main Raspberry Pi menu. Alternatively, press **Ctrl** + **Alt** + **Space** to automatically install Orca.

When booting Raspberry Pi OS for the first time after installing a new image, an automatic spoken reminder plays after 30 seconds. This reminder provides instructions on how to install Orca.

Use Python on a Raspberry Pi

Edit this on GitHub

Raspberry Pi OS comes with Python 3 pre-installed. Interfering with the system Python installation can cause problems for your operating system. When you install third-party Python libraries, always use the correct package-management tools.

On Linux, you can install `python` dependencies in two ways:

- use `apt` to install pre-configured system packages

- use `pip` to install libraries using Python's dependency manager *in a virtual environment*

IMPORTANT

> Starting in Raspberry Pi OS *Bookworm*, you can only use `pip` to install into a Python Virtual Environment (`venv`). This change was introduced by the Python community, not by Raspberry Pi: for more information, see PEP 668.

Install Python packages using `apt`

Packages installed via `apt` are packaged specifically for Raspberry Pi OS. These packages usually come pre-compiled, so they install faster. Because `apt` manages dependencies for all packages, installing with this method includes all of the sub-dependencies needed to run the package. And `apt` ensures that you don't break other packages if you uninstall.

For instance, to install the Python 3 library that supports the Raspberry Pi Build HAT, run the following command:

```
$ sudo apt install python3-build-hat
```

To find Python packages distributed with `apt`, use `apt search`. In most cases, Python packages use the prefix `python-` or `python3-`: for instance, you can find the `numpy` package under the name `python3-numpy`.

Install Python libraries using `pip`

Bookworm changes to `pip` installation

In older versions of Raspberry Pi OS, you could install libraries directly into the system version of Python using `pip`. Since Raspberry Pi OS *Bookworm*, users cannot install libraries directly into the system version of Python.

Instead, install libraries into a virtual environment (`venv`). To install a library at the system level for all users, install it with `apt`.

Attempting to install a Python package system-wide outputs an error similar to the following:

```
$ pip install buildhat
error: externally-managed-environment

× This environment is externally managed
╰─> To install Python packages system-wide, try apt install
    python3-xyz, where xyz is the package you are trying to
    install.

    If you wish to install a non-Debian-packaged Python package,
    create a virtual environment using python3 -m venv path/to/venv.
    Then use path/to/venv/bin/python and path/to/venv/bin/pip. Make
    sure you have python3-full installed.

    For more information visit http://rptl.io/venv
```

```
note: If you believe this is a mistake, please contact your Python installation
or OS distribution provider. You can override this, at the risk of breaking you
r Python installation or OS, by passing --break-system-packages.
hint: See PEP 668 for the detailed specification.
```

Python users have long dealt with conflicts between OS package managers like `apt` and Python-specific package management tools like `pip`. These conflicts include both Python-level API incompatibilities and conflicts over file ownership.

Starting in Raspberry Pi OS *Bookworm*, packages installed via `pip` *must be installed into a Python virtual environment* (`venv`). A virtual environment is a container where you can safely install third-party modules so they won't interfere with your system Python.

Use `pip` with virtual environments

To use a virtual environment, create a container to store the environment. There are several ways you can do this depending on how you want to work with Python:

per-project environments | per-user environments

Many users create separate virtual environments for each Python project. Locate the virtual environment in the root folder of each project, typically with a shared name like `env`. Run the following command from the root folder of each project to create a virtual environment configuration folder:

```
$ python -m venv env
```

Before you work on a project, run the following command from the root of the project to start using the virtual environment:

```
$ source env/bin/activate
```

You should then see a prompt similar to the following:

```
(env) $
```

When you finish working on a project, run the following command from any directory to leave the virtual environment:

```
(env) $ deactivate
```

Create a virtual environment

Run the following command to create a virtual environment configuration folder, replacing `<env-name>` with the name you would like to use for the virtual environment (e.g. `env`):

```
$ python -m venv <env-name>
```

TIP

Pass the `--system-site-packages` flag before the folder name to preload all of the currently installed packages in your system Python installation into the virtual environment.

Enter a virtual environment

Then, execute the `bin/activate` script in the virtual environment configuration folder to enter the virtual environment:

```
$ source <env-name>/bin/activate
```

You should then see a prompt similar to the following:

```
(<env-name>) $
```

The `(<env-name>)` command prompt prefix indicates that the current terminal session is in a virtual environment named `<env-name>`.

To check that you're in a virtual environment, use `pip list` to view the list of installed packages:

```
(<env-name>) $ pip list
Package    Version
---------- -------
pip        23.0.1
setuptools 66.1.1
```

The list should be much shorter than the list of packages installed in your system Python. You can now safely install packages with `pip`. Any packages you install with `pip` while in a virtual environment only install to that virtual environment. In a virtual environment, the `python` or `python3` commands automatically use the virtual environment's version of Python and installed packages instead of the system Python.

Exit a virtual environment

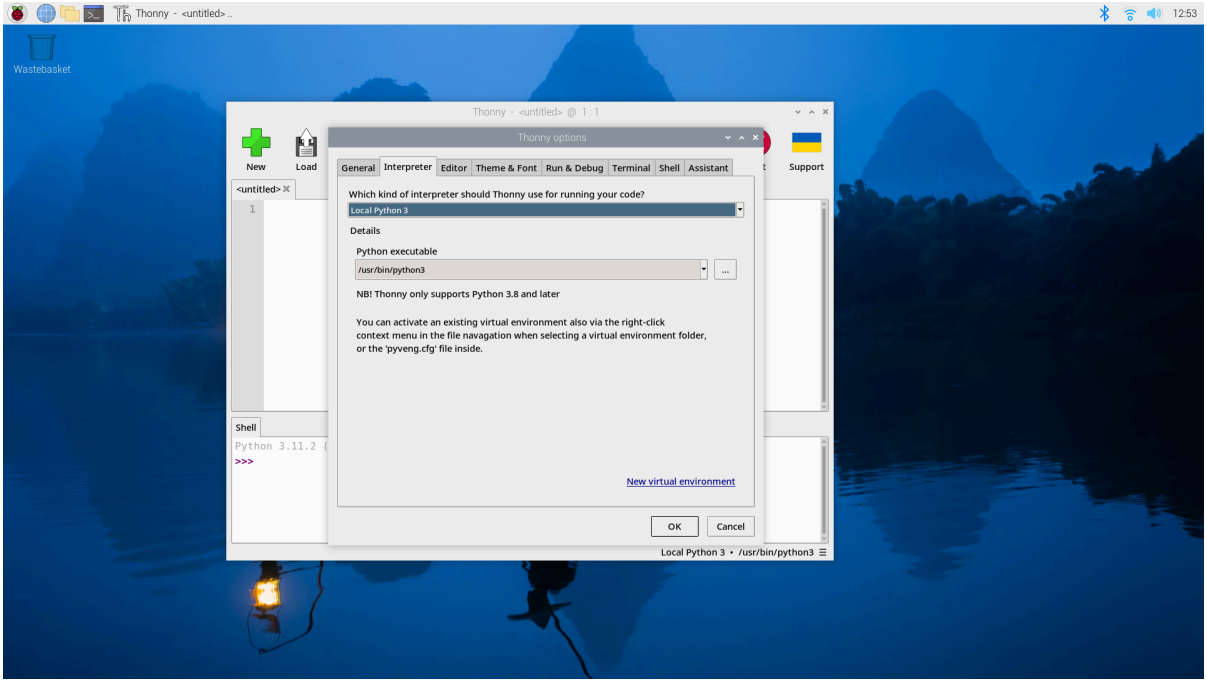
To leave a virtual environment, run the following command:

```
(<env-name>) $ deactivate
```

Use the Thonny editor

We recommend Thonny for editing Python code on the Raspberry Pi.

By default, Thonny uses the system Python. However, you can switch to using a Python virtual environment by clicking on the **interpreter menu** in the bottom right of the Thonny window. Select a configured environment or configure a new virtual environment with `Configure interpreter...`.



Use GPIO from Python

Edit this on GitHub

Using the GPIO Zero library makes it easy to control GPIO devices with Python. The library is comprehensively documented at gpiozero.readthedocs.io.

For information about GPIO hardware, see GPIO hardware.

LED control

The following example code controls an LED connected to GPIO17:

```
from gpiozero import LED
from time import sleep

led = LED(17)

while True:
    led.on()
    sleep(1)
    led.off()
    sleep(1)
```

Run this in an IDE like Thonny, and the LED will blink on and off repeatedly.

LED methods include `on()`, `off()`, `toggle()`, and `blink()`.

Read button state

The following example code reads the state of a button connected to GPIO2:

```
from gpiozero import Button
from time import sleep
```

```
button = Button(2)

while True:
    if button.is_pressed:
        print("Pressed")
    else:
        print("Released")
    sleep(1)
```

Button functionality includes the properties `is_pressed` and `is_held`; callbacks `when_pressed`, `when_released`, and `when_held`; and methods `wait_for_press()` and `wait_for_release`.

Control an LED with a button

The following example code reads the state of a button connected to GPIO2, and lights an LED connected to GPIO17 when the button is pressed:

```
from gpiozero import LED, Button

led = LED(17)
button = Button(2)

while True:
    if button.is_pressed:
        led.on()
    else:
        led.off()
```

Alternatively:

```
from gpiozero import LED, Button

led = LED(17)
button = Button(2)

while True:
    button.wait_for_press()
    led.on()
    button.wait_for_release()
    led.off()
```

or:

```
from gpiozero import LED, Button

led = LED(17)
button = Button(2)

button.when_pressed = led.on
button.when_released = led.off
```

Going further

You can find more information on how to program electronics connected to your Raspberry Pi with the GPIO Zero Python library in the Raspberry Pi Press book Simple Electronics with GPIO Zero. The book gets you started with the GPIO Zero library, and walks you through how to use it by building a series of projects.

You can download this book as a PDF file for free, it has been released under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY NC-SA) licence.