Community Experience Distilled

Learning Banana Pi

Unleash the power of Banana Pi and use it for home automation, games, and various practical applications

Daniel Blair

[PACKT] open source*
PUBLISHING
community experience distilled

Learning Banana Pi

Unleash the power of Banana Pi and use it for home automation, games, and various practical applications

Daniel Blair

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

Learning Banana Pi

First published: June 2015

Production reference: 1250615

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78528-930-9

`www.packtpub.com`

Credits

Author
Daniel Blair

Reviewers
Ryad El-Dajani
Marcel Lange
Martin Lützelberger
Igor Pečovnik

Commissioning Editor
Neil Alexander

Acquisition Editor
Reshma Raman

Content Development Editor
Nikhil Potdukhe

Technical Editor
Edwin Moses

Copy Editors
Karuna Narayanan
Vedangi Narvekar

Project Coordinator
Vijay Kushlani

Proofreader
Safis Editing

Indexer
Hemangini Bari

Production Coordinator
Nitesh Thakur

Cover Work
Nitesh Thakur

About the Author

Daniel Blair is a tech entrepreneur and technologist from Winnipeg, Manitoba. He is heavily involved in start-up and maker communities in Winnipeg and regularly attends and organizes community events that are centered around development and open hardware.

When the Banana Pi board was first released to the community, Daniel got involved in the Banana Pi due to his experience with similar boards, such as the Raspberry Pi. He has worked on various projects, which includes bringing the Tor browser to the Banana Pi.

He has used nearly every incarnation of the Banana Pi and the boards that it has inspired. He has been a part of all kinds of projects related to software and hardware. Although Daniel has written a lot on the Banana Pi on his various websites, this is his first book on the subject.

Bit Space Development, a tech start-up founded by Daniel, offers various solutions to small businesses that need technology such as menu boards, Internet access points, and company dashboards. The power of open hardware has enabled them to offer top-quality support and robust hardware solutions as a small company.

I would like to thank my beautiful girlfriend, Erin, and my amazing daughter, Emily, for putting up with my long writing sessions and procrastination. I would also like to thank the good folks at Bold Innovation Group in Winnipeg for their encouragement and inspiration while writing this book.

I would also like to thank my mentors—Guy Dugas, Alex Trauzzi, and Yvan Boisjoli—for keeping me in line and giving me the support and guidance that I needed to work toward the goals that I have achieved so far with regard to both this book and my career.

About the Reviewers

Ryad El-Dajani is a software engineer and a passionate technology enthusiast. He first became interested in computers when he was 10 years old. Soon thereafter, he began to learn various programming languages.

After he completed his training as an IT specialist, he worked on several e-commerce projects. Today, Ryad El-Dajani studies business computing and works at a big IT company in Germany, where he is a part of various IT projects that are based on Java or .NET technologies. Furthermore, he is authoring *Banana Pi Cookbook*, which will be released by Packt Publishing soon.

Besides the classic application development, he has professional experience in web frameworks such as Spring, Play, Symfony, eZ Publish, and Magento. Moreover, he has been interested in all kinds of Unix-like embedded systems since the revolutionary SheevaPlug.

Marcel Lange is a full-stack web developer from Hamburg, Germany, who specializes in the development of PHP-based e-commerce applications. He holds a degree in applied computer science.

He spends his leisure time experimenting with different kinds of electronics, from soldering LED watches to building home automation systems and remotely accessible robots. The rest of his spare time is devoted to working on his web blog, `http://ask-sheldon.com`, where he shares his knowledge and experience with various types of technologies and programming languages in the form of really short articles or code snippets.

I would like to thank my family and friends for putting up with my rare availability and sometimes brusque answers to e-mails and messages. Thanks for being so lenient with me.

Furthermore, I want to express my gratitude to Packt Publishing for giving me a chance to work on this very interesting project. It gave me many new ideas and experiences. Thanks a lot!

Martin Lützelberger spent many years on biomedical and clinical research in the field of molecular genetics, bioinformatics, and data management. As a long-time Linux enthusiast, he developed a strong interest in microelectronics and physical computing. After much tinkering with Arduino, the Raspberry Pi, and the Banana Pi, he started to work as a teacher of informatics at a high school and brought these single-board computers into the classroom to teach the youth of today how to program and make them understand how computers work.

Igor Pečovnik has been acquainted with electronics since his childhood, which spanned the early eighties. Transistors, tubes, ICs, and the first microcomputer were his devices. He fell into Ham Radio and spent many evenings talking to people all over the world long before the existence of the Internet. During his studies in the mid-nineties, he fell in love with Linux. In this period, he was involved in the mainstream Internet infrastructure deployment. He worked on various hardware and software projects and started his own web publishing company, where he spent a decade working as a CEO. The company operated one of the biggest local forums. He took a break from entrepreneurship for a while and went back to intensive humanistic study for years and focused on his family. After he became a father of two, he went back to his first love—electronics. Once, he bought a small developer board and figured out that its operating system was crappy. This motivated him to start developing his own OS. The Debian Linux operating system, together with an open source build kit, is most popular for CubieBoard, Banana PI, and Olimex Lime, to name a few. He is a recognized member of the communities of these small boards. After devoting a few years toward the development of systems, he still develops and supports the basic Debian Linux for a dozen ARM boards.

I am working on a public project per se, so I have many coworkers whom I want to thank. Thanks to all those who use, test, and report problems, the ones who gave compliments and motivation, and the people who donate money and devices!

Table of Contents

Preface

This book covers the topics of the Banana Pi and single-board computing. Single-board computing generally refers to the computers that reside entirely on one chip. The computer that we are dealing with is the Banana Pi, which is based on the Raspberry Pi, the open source computer.

Single-board computing has become extremely popular as of late for its use in prototyping, education, and gaming. This book will introduce you to all of these topics. We will go from the beginner to the intermediate level in just a few chapters.

The communities that developed these devices have come up with many cool use cases, such as tiny low power servers, media devices, and arcade machines. We will explore a few popular uses of the Banana Pi, which will show you just how versatile your little computer is.

Also, there are a lot of different operating systems that are available for the Banana Pi. We will explore the set up and use of two of the most popular operating systems, Android and Linux. You will be an intermediate user by the time we are done here, which will set us up to explore complex projects, such as robotics, further down the road.

What this book covers

Chapter 1, *An Overview of the Banana Pi*, gives an introduction to the Banana Pi board. It also covers the basic components of the Banana Pi.

Chapter 2, *Preparing the SD Card*, covers one of the most important steps to owning your first Banana Pi. We will learn how to prepare the SD card on multiple operating systems.

Chapter 3, *Linux and the Command Line*, talks about the Linux operating system and some useful commands. This is important when you are using the Banana Pi because almost all of the operating systems for the Banana Pi are based on Linux.

Chapter 4, *Programming on the Pi*, explores the different programming languages that are supported by the Banana Pi. This will introduce the user to some code and different use cases.

Chapter 5, *Hardware for Your Pi*, explores some of the hardware that you can use with the Banana Pi. We will explore electrical components such as the LED and capacitor.

Chapter 6, *Interacting with Sensors*, deals with sensors, which are very useful in circuits. We will have a look at a few different sensors and how they can be used.

Chapter 7, *Building an Internet Radio*, is built upon the discussions from the previous chapters and shows you how to construct a media device. The Internet radio will be explored in a couple of ways, which will allow for it to be used more easily.

Chapter 8, *Building a Home Server*, takes a look at how you can build a low-power, low-cost Banana Pi home server. We will delve into the Google Apps replacements and shared calendars. All of this will be powered by your own hardware.

Chapter 9, *Gaming on Your Pi*, is the most fun chapter of this book. We will explore video games on the Banana Pi. We will look at building a retro game computer and explore game streaming on Android.

What you need for this book

This book will require you to have access to another computer other than your Banana Pi. You will need to download some specific software, such as the Phoenix card writer, to prepare the Android image. Furthermore, you will need access to putty to SSH into the Banana Pi on Windows.

Besides the aforementioned requirements, you will just need access to your favorite text editor, a keyboard, and some basic electrical components, such as the breadboard, resistors, and LEDs.

Who this book is for

This book is for the users who will be using the Banana Pi for the first time or are new to single-board computing. We will learn a lot in a short amount of time. So, be prepared for a fast-paced journey into the world of the Banana Pi.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Once you have done this, you will need to run `fdisk` on that particular drive."

A block of code is set as follows:

```
#!/usr/bin/env python
import RPi.GPIO as GPIO

#Use BCM numbering
GPIO.setmode(GPIO.BCM)
```

Any command-line input or output is written as follows:

```
sudo fdisk /dev/sdx
```

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "We can enter this IP into the **Host Name** input field on PuTTY"

[Warnings or important notes appear in a box like this.]

[Tips and tricks appear like this.]

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from `http://www.packtpub.com/sites/default/files/downloads/9309OT_ColorImages.pdf`.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

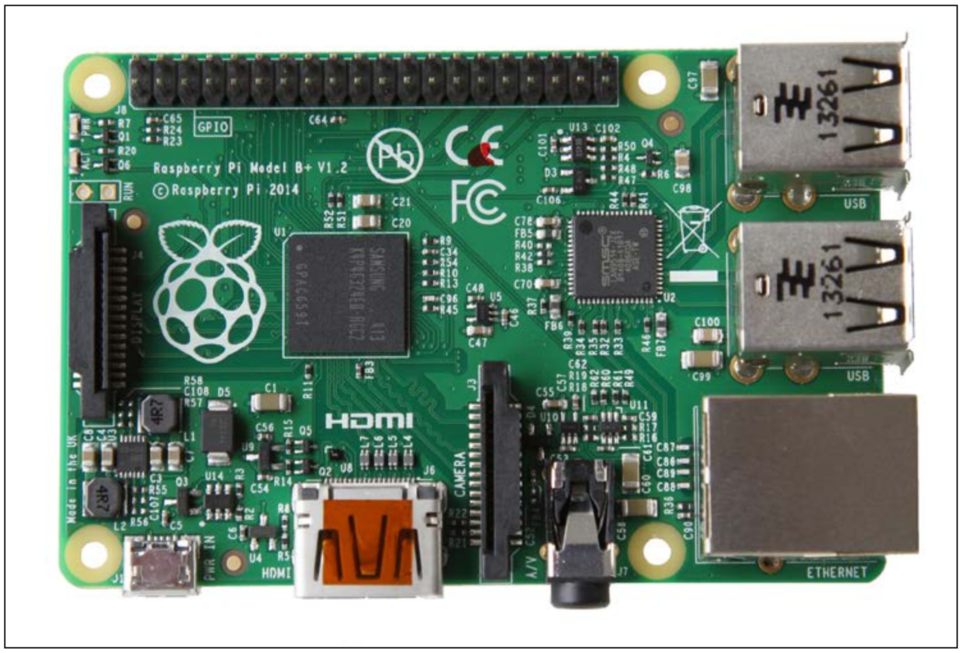
We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

1
An Overview of the Banana Pi

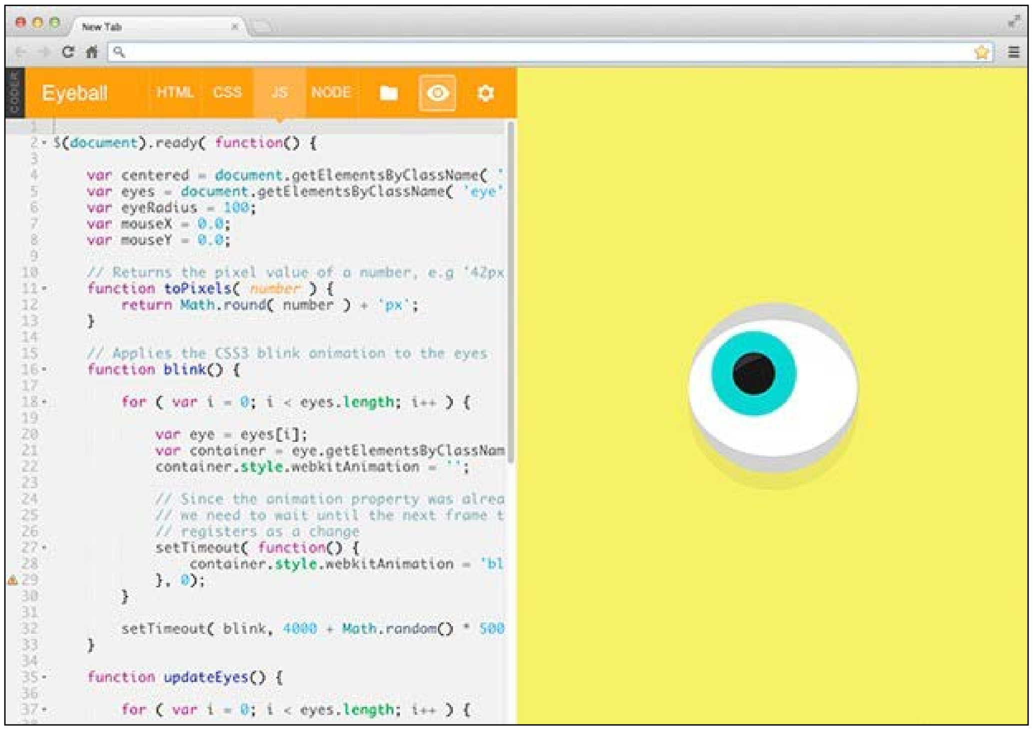
This book is going to help you on your journey into single-board computing. We are going to cover a lot of topics that are popular in the community and build some skills that will help us when we are working on projects later on. The world of single-board computing is new and fast-paced but it opens up so many doors when it comes to education and technology. The Raspberry Pi (pictured here) is the inspiration for the Banana Pi.



Single-board computing is a relatively new technology that was popularized by the Raspberry Pi and really made it easy to prototype and learn. The Raspberry Pi brought cheap computing power to people around the world and empowered kids and hobbyists to learn about Linux and start to play with electronics. It brought out the maker in all of us and we started to build and see what we could do.

The Banana Pi allows you to take rapid prototyping to the next level. With an operating system directly on the board and access to the **general purpose input output** (**GPIO**) pins, you can not only set up a **home theatre PC** (**HTPC**), you can actually build the next great product using one.

Learning electronics is easier than ever when you attach the GPIO output to a breadboard and build directly from that. The Banana Pi, like the Raspberry Pi, is so small you can even incorporate it into a finished product and forego the need to implement another micro controller. The GPIO pins allow you to write code on the device that interacts with physical hardware such as LEDs, resistors, and switches. This means you can mock-up your idea, and get a working version of it before making a commitment to manufacturing something.

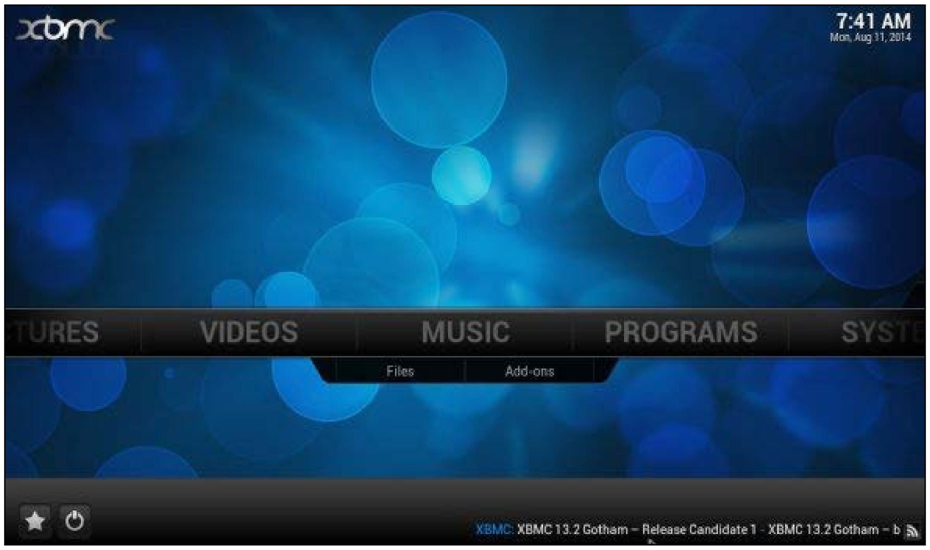


```

$(document).ready( function() {

    var centered = document.getElementsByClassName( '
    var eyes = document.getElementsByClassName( 'eye'
    var eyeRadius = 100;
    var mouseX = 0.0;
    var mouseY = 0.0;

    // Returns the pixel value of a number, e.g '42px
    function toPixels( number ) {
        return Math.round( number ) + 'px';
    }

    // Applies the CSS3 blink animation to the eyes
    function blink() {

        for ( var i = 0; i < eyes.length; i++ ) {

            var eye = eyes[i];
            var container = eye.getElementsByClassNam
            container.style.webkitAnimation = '';

            // Since the animation property was alrea
            // we need to wait until the next frame t
            // registers as a change
            setTimeout( function() {
                container.style.webkitAnimation = 'bl
            }, 0);
        }

        setTimeout( blink, 4000 + Math.random() * 500
    }

    function updateEyes() {

        for ( var i = 0; i < eyes.length; i++ ) {
```

Coding is something that is becoming very important as a skill now, and with coding tutorials, and software that you can install directly on the Banana Pi, you can build skills that will be valuable to you for a long time. The Google Coder interface is pictured in the preceding screenshot.

The Banana Pi is usable as a web server, which means that you can learn languages such as Ruby, PHP, and Python directly on the board, or push code from your other computer to the Pi, which can act as your local development server. I have found projects such as Google Coder to be incredibly useful tools for learning these skills.

Popular development tools are available for the Banana Pi on Linux as long as there is an ARM-compatible build for it. This includes software such as gedit. You can also run web-based solutions (for instance, Adafruit's WebIDE, which runs as a web service directly on the Pi).



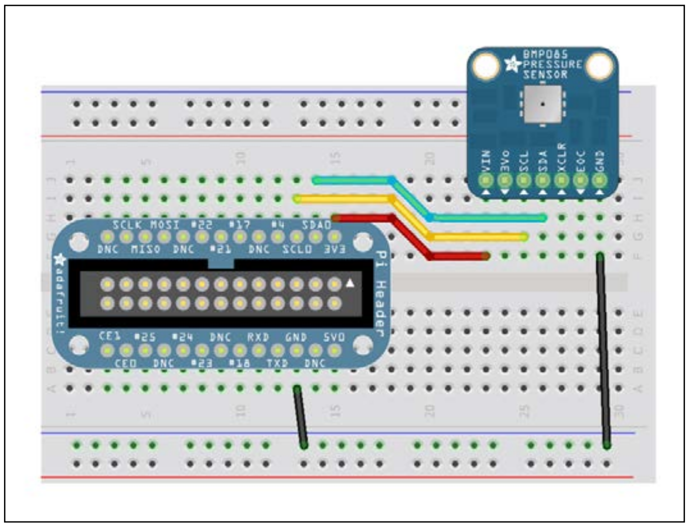
One of the most popular uses of the Raspberry Pi is as an HTPC or Media Centre PC. This means that the device is hooked up to a TV and used for the displaying of media. With tools such as XBMC (see the preceding screenshot) and Samba for file sharing, this is quite easy. The Banana Pi makes this even more accessible. What used to be a slower GUI is now sleek and fast and, with the built-in IR receiver, you can even use a remote to control the Pi from the sofa without the need for a keyboard and mouse. The SATA port on the M1 makes it faster than ever to get media to your TV due to the read/write speed of SATA being a lot faster than USB. The M2's built-in Wi-Fi is also very useful for this task.

Popular software such as Octoprint (used to control a 3D Printer—for example, the Printrbot) is also compatible with the Banana Pi. This allows you to directly connect your Banana Pi to your 3D printer and use that as a print server. You can send your 3D models to the Pi and have it slice up your model and start the print. This is one of my favorite uses for the Pi, something that also translates into rapid prototyping.

Differences between the Raspberry Pi and Banana Pi

The Raspberry Pi took open source software and added a hardware aspect to it. Now for less than $100 you could have a cheap computer and that really got people excited. A huge community has grown from the Raspberry Pi, which has helped with the success of the board and has been the inspiration for newer boards with better specs such as the Banana Pi. The Banana Pi is a board that builds on the success of the Raspberry Pi. It is a single-board computer, which means that everything needed to operate on the system, is built-in, all you need to do is add your peripherals. The Banana Pi takes it a step further with increased stats and performance, which allow you to really take your projects to the next level.

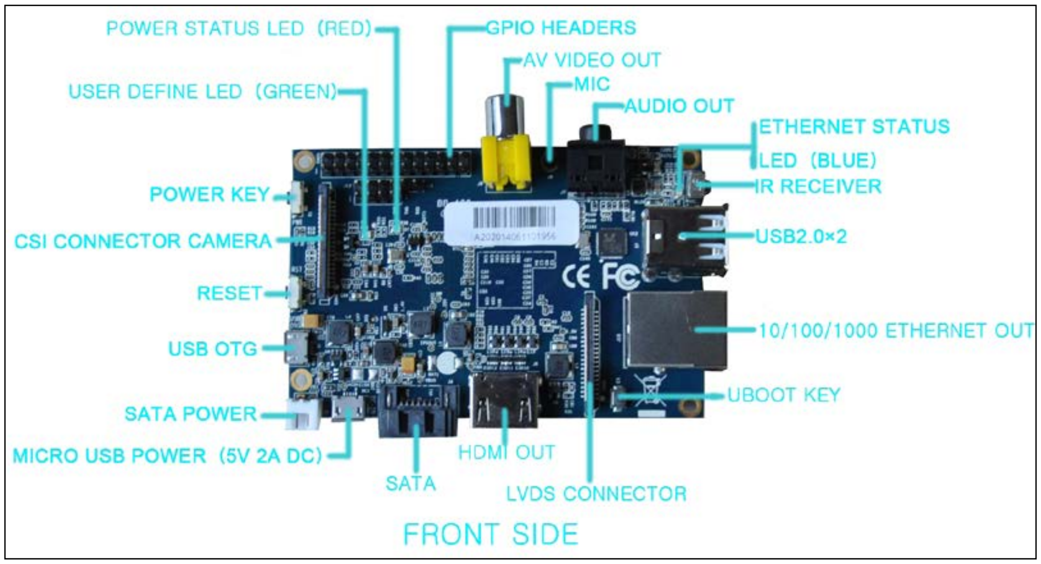
The community for the Banana Pi is growing and you can already find projects and tutorials specific to it. The additional hardware such as the SATA port and Gigabit LAN has made it very popular for file servers or ownCloud. The nice thing is, you can still do everything you wanted to do with the Raspberry Pi, but with a little extra hardware. The following screenshot shows some hardware used to breadboard a Banana Pi.

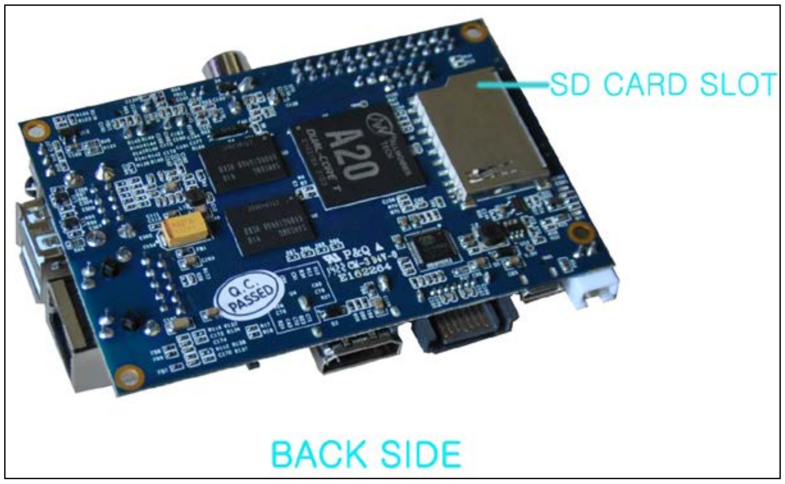


The M1

There are two versions of the Banana Pi: the M1 and the M2. The M1 is more common and currently sports a 1.2 GHz dual core processor, while the M2 is newer and has a 1.4 GHz quad core processor. Both boards operate similarly, but the M2 is really where things are headed. You can identify the board you are using by counting the GPIO pins in the bottom-left corner of the board. If you have 40 (like the Raspberry Pi B+) then you have the M2. If you have 26, like the classic Raspberry Pi, then you have the Banana Pi M1.

Getting familiar with the hardware on the Banana Pi is important; this will help you determine what can be done with the Pi, and will help you decide which model to use in your future projects at the planning phase. The M1 board has a very similar layout to the Raspberry Pi model B (classic). On the left side of the board you have your 26 GPIO pins, AV video out, and 3.5 mm audio out. The front of the board has your LED indicators, IR receiver, and Ethernet port. The ports for the M1 are outlined in the following diagrams:





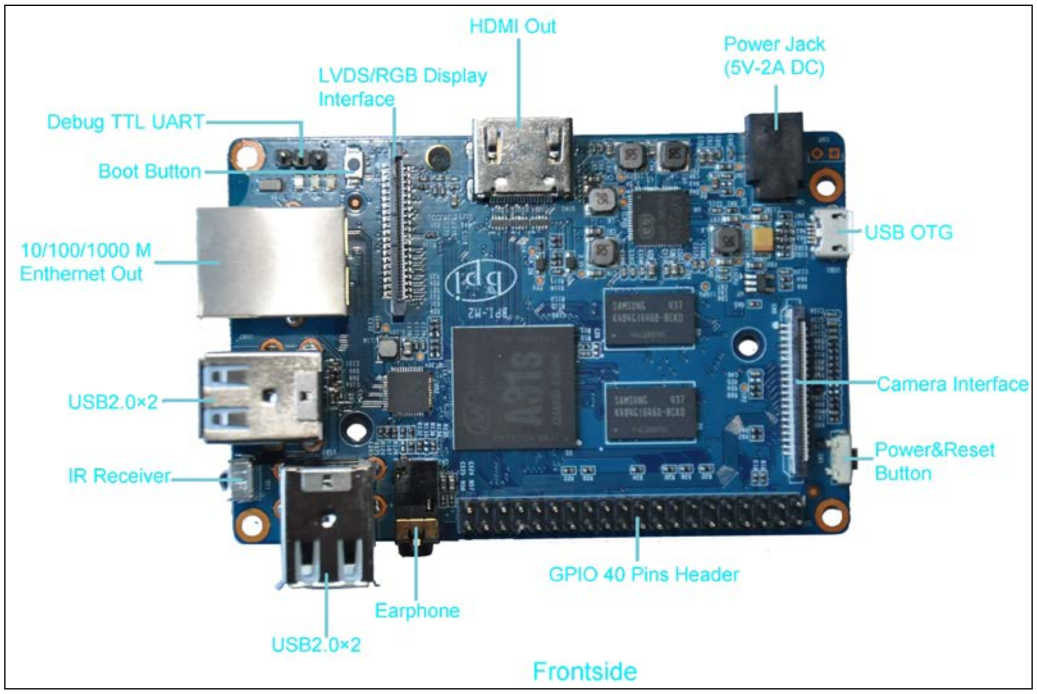
Both the M1 and M2 boards have nearly identical mounting holes around the four corners of the board.

The M2

Down the right side of the board you have the UBOOT key, LVDS connector for video displays, HDMI out, SATA, 5 v micro USB power port, and SATA power out. The bottom of the board has the power and reset switches, as well as a micro USB OTG (host) port. The back side of the board shows the CPU and the SD card slot for the OS.

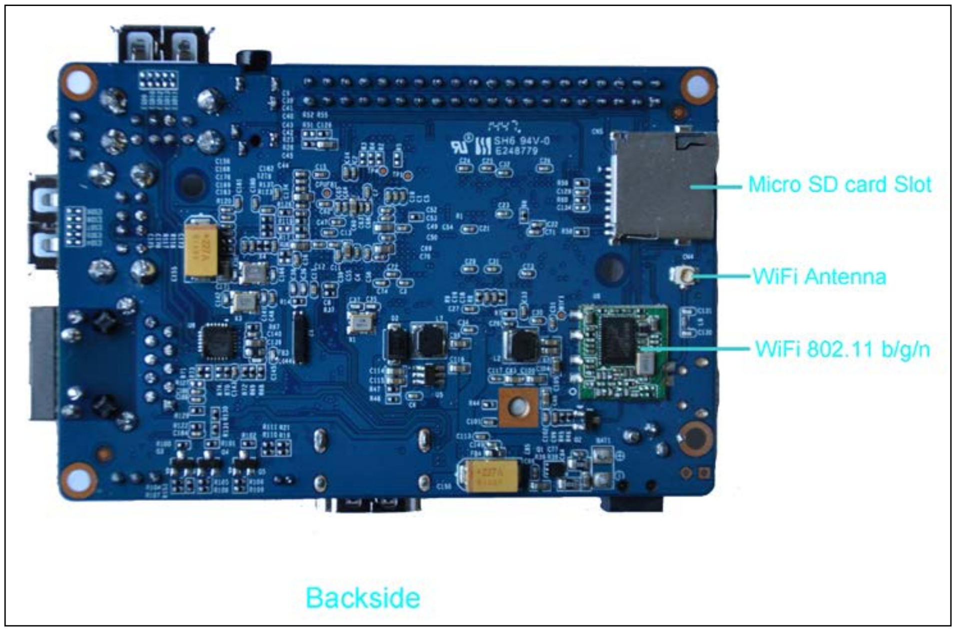
The M2 board is similar in layout to the Raspberry Pi model B+, with 40 GPIO pins in the bottom-left corner of the board. The layout does differ though. On the left-hand side of the board you will find the GPIO pins, 3.5 mm audio jack, and two USB 2.0 ports. On the top of the board you will find the IR receiver, two more USB 2.0 ports, and the Ethernet port. The right-hand side of the board has the boot button, LVDS/RGB display interface, HDMI out, and 5V, 2A power jack (not USB). The bottom edge of the board has the camera interface, power and reset buttons, and the micro USB OTG host port.

These ports are outlined in the following diagram:



The bottom side of the board has the CPU, Wi-Fi antenna connector, RealTek 802.11 b/g/n Wi-Fi adapter, and the micro SD card slot for the OS. The biggest difference between the M1 and the M2 is the lack of SATA when upgrading the M2. This may determine which model you plan to use, or it may not affect your decision. The M2 supports Micro SD and Wi-Fi, unlike the M1, which does impact the final project if those are features you are looking for.

The next image shows the bottom side of the M2:



Overall, the boards are similar in terms of architecture and ports with several exceptions for each board. You could compare the Banana Pi with the Model B versions of the Raspberry Pi. The Raspberry Pi Model B+ builds upon and improves some aspects of the Model B classic, however both are still useful. The M1 is not necessarily worse than the M2, since it offers different hardware and even some different software to the M2. The following chart gives a direct comparison of features between both versions of the Banana Pi. The Banana Pi M1 uses the A20 processor, which is currently very widespread and widely supported. The A31 boasted by the M2 is still very new, and the lack of SATA on the board is a shortcoming of the board's design.

The following table outlines the differences in hardware between the Banana Pi M1 and M2. The boards are comparable; although the M2 is the upgrade, the M1 has features that the M2 does not.

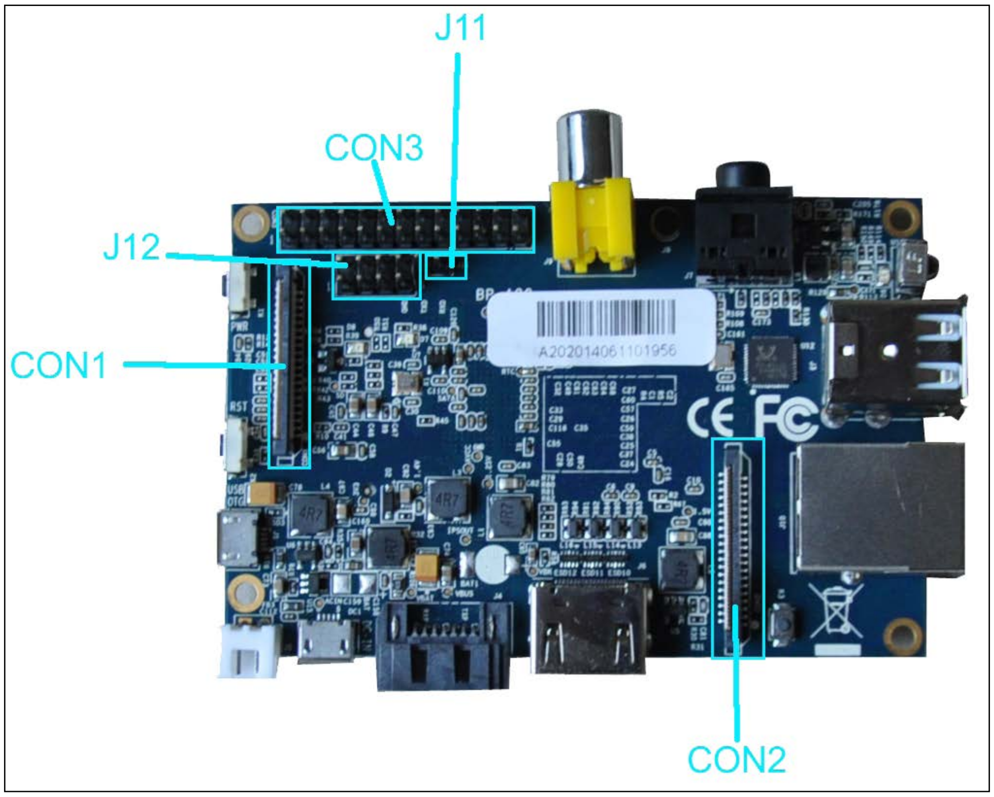
The following table visualizes the differences between both the M1 and M2 Banana Pi boards:

	Banana Pi M1	**Banana Pi M2**
CPU	A20 ARM® Cortex™-A7 Dual-Core	A31S ARM Cortex-A7 quad-core, 256 KB L1 cache 1 MB L2 cache
GPU	ARM Mali-400MP2 complies with OpenGL ES 2.0/1.1	PowerVR SGX544MP2
Memory (SDRAM)	1 GB DDR3 (shared with GPU)	1 GB DDR3 (shared with GPU)
Storage	SD max. 64 GB	Micro SD max. 64 GB
Ethernet	10/100/1000 Ethernet RJ45	10/100/1000 Ethernet RJ45
Wi-Fi	N/A	802.11 b/g/n
Video Outputs	HDMI, CVBS, LVDS/RGB	HDMI, CVBS, LVDS/RGB
Audio Output	3.5 mm Jack and HDMI	3.5 mm Jack and HDMI
Power Source	5 volt via MicroUSB (DC In Only) and/or MicroUSB (OTG)	5 volt via MicroUSB (DC In Only) and/or MicroUSB (OTG)
USB 2.0 Ports	2x	4x
IR Receiver	Yes	Yes
Buttons	Power, Reset, UBoot	Power, UBoot

GPIO pins

The **general purpose input output** (**GPIO**) pins are one of the best features of both the Raspberry and Banana Pi boards. These pins are what allow your software to interact with the physical world much like working with an Arduino. These pins can be interfaced with, via code right from within the operating system running on the board. The pins can be used to interface with sensors, displays, touch screens, and the layout is exactly the same as the Raspberry Pi; thus, basically all the accessories are compatible with the Banana Pi.

The next diagram shows the location of all the GPIO pins:



CON1 is the extensible on-board CSI connector. It is a 40-pin FPC connector that can be used to connect the external camera module. CON2 is an LVDS display connector. CON3 contains the CAN bus, SPI bus, I2C bus, PWM, serial port, and so on. It can be configured to be used for all kinds of peripherals. The table viewable from the link contains the CON3 pin definitions. This is a reference for future use as you will need to know which pin is used for what.

CON3 GPIO pin definitions

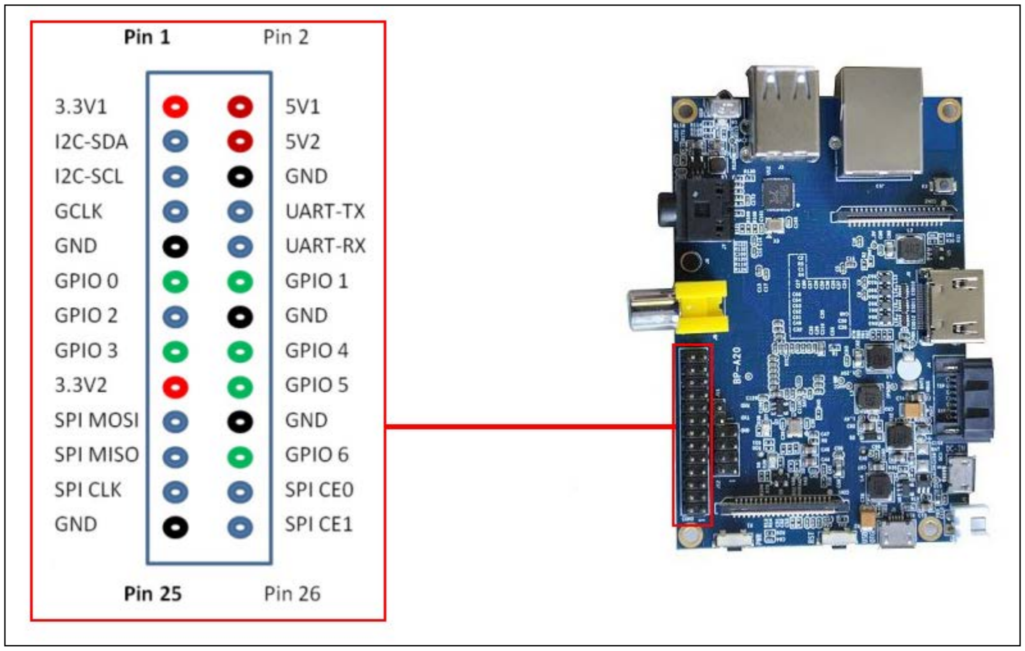
The definitions for the GPIO pins can be located on the official website. These are quick and easy references and will benefit you when working with the Banana Pi. This following link will get you to the tables: `http://www.bananapi.com/index.php/component/content/article?layout=edit&id=24`.

J11 contains the default serial port. J12 also contains a serial port UART. UARTO (UARTO-RX, UARTO-TX) is configured for console input/output. This is useful if you want to log in using the serial port, which makes it the most commonly used PIN.

UART stands for Universal Asynchronous Receiver/Transmitter. This is hardware that translates data between serial and parallel sources. This means that those pins are directly connected to a piece of hardware that translates this data. This is used for directly connecting to the board and diagnostics, among other things.

The GPIO pins represent all of the interfaces you may need while working with the Banana Pi. This includes connecting over serial, and even powering the board. These pins allow you to interact with physical components through code you have written on the board. This means you can, for example, signal an alarm when a moisture sensor detects that your plants are dry, or write a row in your database when your beer has reached a certain stage while brewing.

GPIO is not specific to the Raspberry or Banana Pi; this represents any general pins that are on a board. This means that these pins are merely input and output and are there for the sake of allowing you to work with them. The following image shows the GPIO pins in relation to the actual device:



Operating systems

The Banana Pi supports many different operating systems. Because of its ARM v7 processor, it also runs operating systems such as Android. The list is actually quite long but currently there are several ported Raspberry Pi builds of Linux, which includes Kano and Raspbian as well as an optimized build called Bananian. There are also several ARM-based Linux builds targeted at the Banana Pi to choose from. These builds include OpenSUSE, Arch, Fedora, Lubuntu, and even Kali Linux. There are also two versions of Android available: 4.2.2 Jelly Bean and 4.4 KitKat.

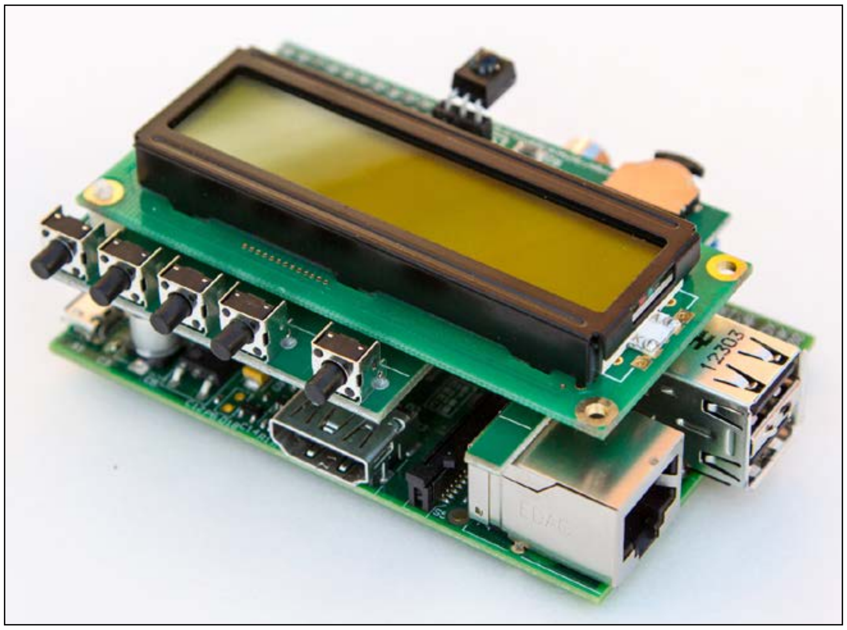
The choice in operating systems is part of the reason the Banana Pi is so versatile. The operating system you choose may impact your project; something like Android would make a great media PC, whereas Arch or Debian would make a better file server. The choice is there though and all are available for free from the official website at `http://www.bananapi.com/index.php/download`.

The most common choice in operating systems for the Banana Pi is the Bananian port of Raspbian. This is due to the popularity of Raspbian for the Raspberry Pi. Basically every Raspbian-based project for the Raspberry Pi is directly compatible with the Banana Pi. This means that, by sticking with Bananian or Raspbian, you immediately have access to loads of tutorials and projects. Raspbian Linux is based on Debian Linux; this is one of the most popular distributions in the world.

Raspberry Pi accessories

The Banana Pi is made to be compatible with Raspberry Pi accessories. This is of course in relation to boards that are attached directly.

The GPIO layout of the M1 Banana Pi is identical to the Raspberry Pi Model A and B (classic). That means that boards such as the PiFace command and control will directly fit on the board and function just as if it was connected to a Raspberry Pi. The Banana Pi boards have a different layout than the Raspberry Pi boards, which means that you can't use your Raspberry Pi cases for the Banana Pi and vice versa.



Getting started

Now that you know about the hardware aspects of the different Banana Pi boards and their differences from, and compatibilities with, Raspberry Pi boards, you can start to plan out your projects.

Setting up your environment

There are a number of ways that you can set up your actual work environment for working with the Banana Pi. Depending on the board you are using, there are various cables that you will need to get started, some of which can later be removed depending on the project you are working on.

As you read earlier, the different boards have different layouts. There are some common cables that you will need to get started. Both versions of the Banana Pi have an HDMI out interface. Generally, no matter what the project, you are going to connect it via HDMI even if it is just to get the project started.

Both boards can be powered over USB, although the M2 does have a 5V power jack slot that can be used to power the board as well. The USB cable needed to power the Banana Pi is a standard Micro USB cable that is similar to, if not exactly the same as, the one you may use to charge your cell phone. You can connect a USB keyboard and mouse to either board, and of course you will need an SD card to hold the operating system. The essential hardware you are going to need to work with the Banana Pi is:

- HDMI cable
- Ethernet cable
- Wi-Fi dongle for the M1 (the M2 has built-in Wi-Fi)
- USB keyboard and mouse
- SD card (Micro for M2)

Once you have set up the initial board, you will be able to downsize some of this hardware. If you plan on using SSH to connect to the board, you won't need a display, keyboard, or mouse, which leaves you with a very small footprint.

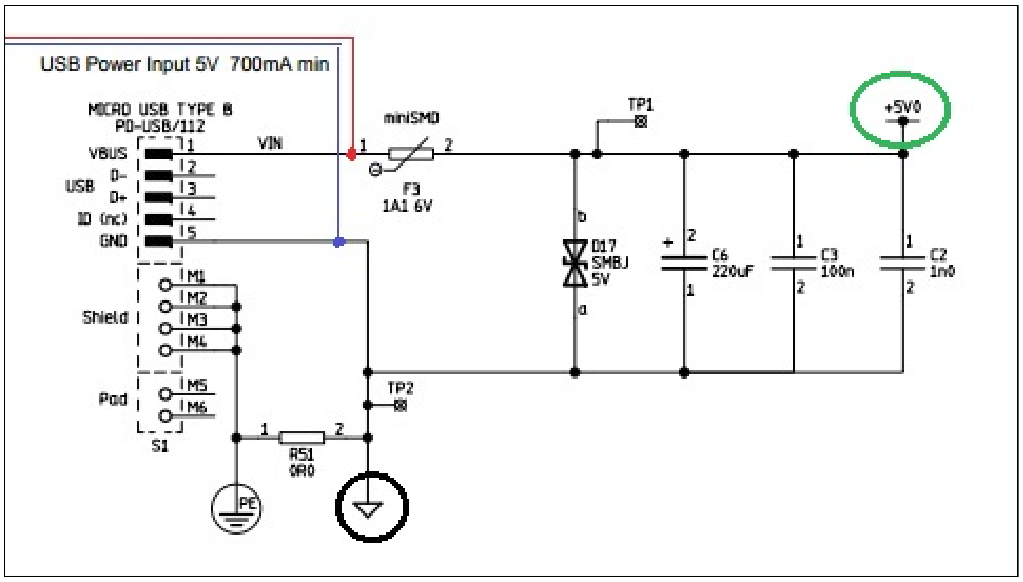
Powering the boards

There is more than one way to power the boards. In fact, there are three ways:

- Micro USB through the OTG Port
- 5V connector for the M2
- Power through the GPIO

The most common way to power the board is through the micro USB port. This is usually because we all seem to have a surplus of cables lying around and they generally tend to be USB from hardware such as cell phones and controllers. The M2 also comes equipped with a 5V DC connector that is new to the Banana Pi.

One way you can power your project that may be right for you is over GPIO. The pins are connected to the 5V rail, which means you can supply power to the pins. This means you can wire up whatever sensors and electronics you need for your project, as well as a direct power line. This is useful if you are planning on doing something battery- or solar-powered. The +5V supplied from the USB connector is filtered to give a stable 5V supply to the 5V0 Rail.



There are implications to powering the board this way, though. Although you can supply 5V of power though the GPIO pins, there is no protection for the board. This is also true if you use the 3.3V pins. This means that the board is not protected from spikes and you run the risk of a fried board. It is really recommended that you stick to micro USB or the 5V connector unless you are prepared to build in that protection before supplying power. Never power the device through GPIO pins AND micro USB at the same time. Always use only one power source.

Summary

With the information we have covered so far you should be able to start doing some projects. From here you will need to prepare your SD card with the operating system of choice and gather the supplies you will need to get started.

You also have the tools necessary to start working with the GPIO pins as well as a layout of the actual pins. This will help you later on when we start to do some interesting stuff with sensors and electronics.

In the next chapter, we are going to cover some of the software, options, and the process of installing the operating system onto the SD card for our Banana Pi. We will cover Windows, Linux, and Mac OS X as the process varies depending on your computer's operating system.

2
Preparing the SD Card

In this chapter, we are going to cover a variety of topics ranging from preparing your card to installing your operating system. We will also cover installing Android and configuring your installation.

As mentioned in the previous chapter, there are many different choices when it comes to your operating system. The most common choices are Android for Media PCs and Bananian or Raspbian, which are both based on Debian Linux. There are several different ways to accomplish preparing your SD card for use on different operating systems and we will cover each of them.

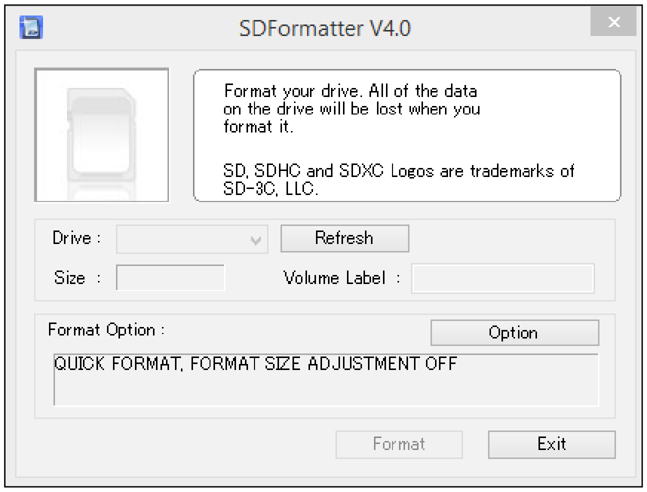
Getting the SD card ready

Regardless of the operating system on your computer, you need to have access to an SD card writer. Most laptops have them; depending on your desktop, you may or may not have a reader but, if you don't, these items are inexpensive and will be necessary for working with your Banana Pi and any other single-board computer that uses SD cards and microSD cards.

If you bought a brand-new SD card (Class 10 preferred) that has been preformatted with FAT32 then you are good to go; in fact some kits come with an operating system preinstalled on the SD card that will let you skip this whole section. If that is the case, keep this section as information in your back pocket if you choose to switch operating systems eventually or want to reinstall later.

Prepping the SD card on Windows

The quickest and easiest way to actually get your card wiped and ready to go, is to use a piece of software called SD Formatter. This tool is distributed by `http://sdcard.org` and is the official tool for working with SD cards. You have only a few options when working with the software, as seen in the following screenshot:

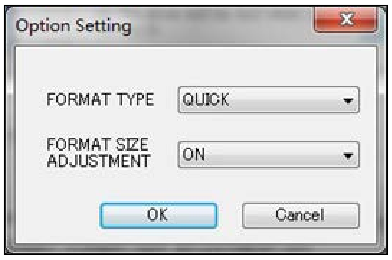


Format your drive. All of the data on the drive will be lost when you format it.

SD, SDHC and SDXC Logos are trademarks of SD-3C, LLC.

Format Option :
QUICK FORMAT, FORMAT SIZE ADJUSTMENT OFF

Once the software is installed and launched, you will be presented with a small window that has a few options. You are asked to choose the drive, which is a drop-down list of the removable devices attached to the computer. You are shown the **Size**, **Volume Label**, and **Format Option** options. You can change these options under the **Option** button where you can choose the format type and format size adjustment as seen in the following screenshot:



FORMAT TYPE: QUICK
FORMAT SIZE ADJUSTMENT: ON

The official Banana Pi website tells you to set your format type to **Quick** and to leave the size adjustment to **On**. I have found that personally, I have a lot more success working with the cards when I set the format type to **Full (Erase)**, which is likely due to the fact that we are installing an operating system. I find that Quick is generally fine for drives you intend to just store files on. A full format checks the disk for bad sectors, which takes longer but will result in a much better write. Once this is all set and you have selected your drive from the drop-down, you can click on **Format** and begin the process. When this is done, you will have a clean card to start working with. Unfortunately if you have already installed Android using Phoenix, you will need to format the disk using `fdisk` or GParted on Linux.

Prepping the SD card on Linux

Since we are going to be working with Linux later on, it makes sense that you can prepare your card from the same OS you will be installing on. I personally prefer working on Linux but that is my preference, I can live on any OS as long as I have a terminal.

Preparing the drive from the command line

There is more than one way to prepare your SD card on Linux and some people tend to lean towards the command line side of things. This is a quick and efficient way to get things done. This is also good practice for later since a lot of working with the Banana Pi is command-line-oriented.

To get started, you are going to need to identify your drive. You may need to be running as root or as a user with root privileges. So, for the sake of this book, we will run our commands with `sudo`. You can run the `fdisk` command to check all of the attached drives:

```
sudo fdisk -l
```

It should be noted that mixing up the device path might lead to severe data loss!

This output will supply you with a list of the drives attached to the computer; you will need to note which one is your SD card. Once you have done this, you will need to run `fdisk` on that particular drive.

```
sudo fdisk /dev/sdx
```

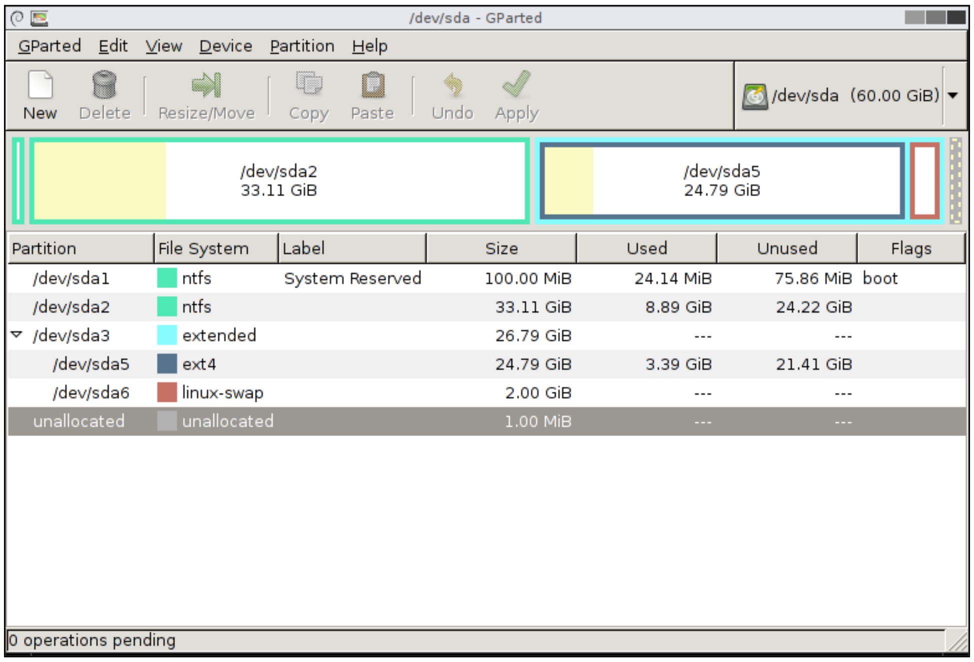
The x character in sdx will be replaced by the drive number for your drive. The most common path for the SD card is /dev/mmcblk0. Once you have run the command, you will have several options. You will need to select d to delete all the partitions on the drive. You will select n to create a new partition and finally w to write the changes. Now you are able to create a filesystem on the empty partition. This is accomplished by running the following command:

```
sudo mkfs.vfat /dev/sdxx
```

On some fresh installs of Linux you may need to install the dosfstools package. The xx character will be replaced according to your drive as seen in the previous case. Once this is finished, you will have an SD card with a clean FAT32 filesystem ready to have an OS installed on it.

Preparing your drive using GParted

If you do a lot of drive maintenance under Linux, you may be familiar with GParted. If not, then this is going to be one of your favorite tools. GParted is a utility for formatting, deleting, and creating drives and partitions. GParted can also be used to create a live CD/USB that you can boot to and accomplish the same task; this also doubles as a valuable tool to have kicking around whenever you need to work with drives. This tool is far more robust than SD Formatter for Windows as you can see in the screenshot.


/dev/sda - GParted
GParted Edit View Device Partition Help
New Delete Resize/Move Copy Paste Undo Apply
/dev/sda (60.00 GiB)
/dev/sda2 33.11 GiB
/dev/sda5 24.79 GiB

Partition	File System	Label	Size	Used	Unused	Flags
/dev/sda1	🟩 ntfs	System Reserved	100.00 MiB	24.14 MiB	75.86 MiB	boot
/dev/sda2	🟩 ntfs		33.11 GiB	8.89 GiB	24.22 GiB	
/dev/sda3	🟦 extended		26.79 GiB	---	---	
/dev/sda5	🟦 ext4		24.79 GiB	3.39 GiB	21.41 GiB	
/dev/sda6	🟥 linux-swap		2.00 GiB	---	---	
unallocated	⬜ unallocated		1.00 MiB	---	---	

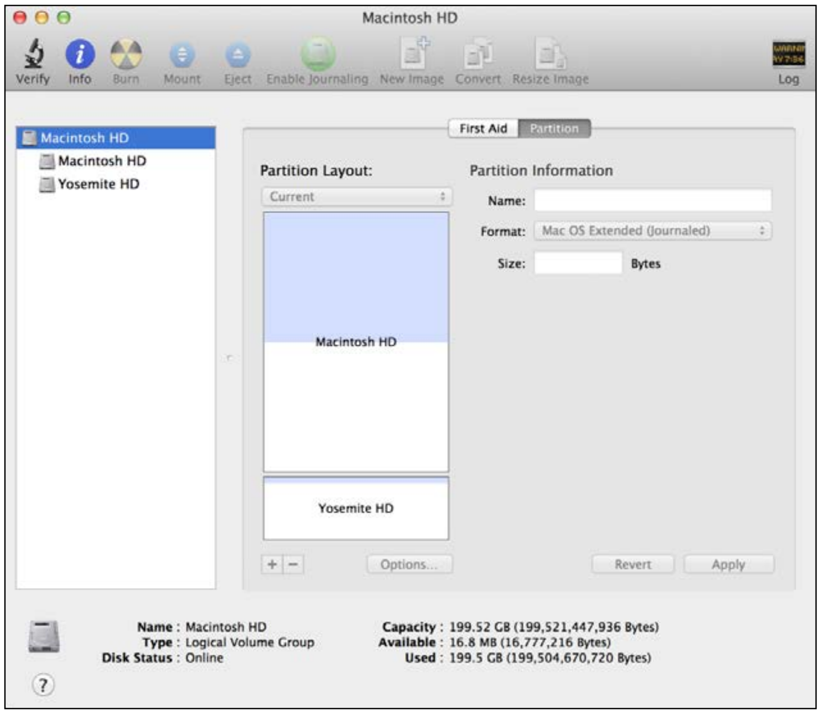
0 operations pending

Some of the features in GParted include creating, deleting, resizing, and moving partitions. GParted is also great for working with filesystems such as FAT12/FAT32, ext2, ext3, and ext4, NTFS, and more. This tool will allow you to delete and create your new partitions.

To actually prepare your card in GParted and get things rolling, all you have to do is select the drive in the top-right corner and unmount any partitions that may be on the drive already. You can accomplish this by right-clicking on them in the list on the graph and selecting **unmount**. Once this is done you will be able to right-click and select **delete** from the context menu and create your new partition. Once the drive is done you can click on the green **Apply** button on the top bar and everything will be written. At this point, you are ready to install your operating system.

Prepping the SD card on the Mac

The Mac OS is actually based on Unix, which means that you have a very similar command-line interface as Linux; that being said, many Mac users are going to be more comfortable using the disk utility to prepare the card. This can be reached by opening the setting page either by searching or by opening it from the overview feature. The utility is a relatively simple looking tool that has a similar functionality to GParted. You can see the UI in the following screenshot:



Name:
Format: Mac OS Extended (Journaled)
Size: Bytes

Name : Macintosh HD
Type : Logical Volume Group
Disk Status : Online

Capacity : 199.52 GB (199,521,447,936 Bytes)
Available : 16.8 MB (16,777,216 Bytes)
Used : 199.5 GB (199,504,670,720 Bytes)

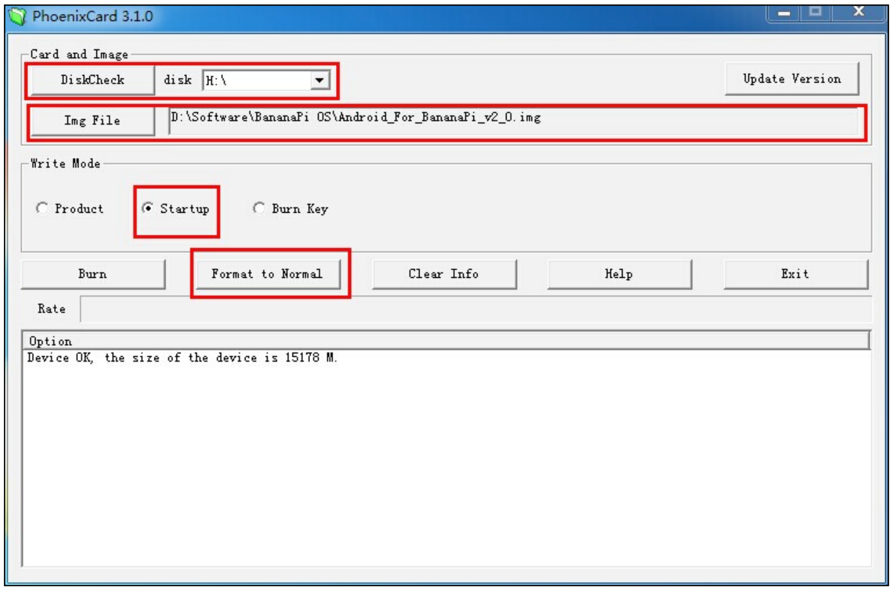
Once you have launched the Disk Utility you will be presented with a list of attached storage devices on the left-hand side. This list will include your main drive, external drives, and your removable media such as USB thumb sticks and SD cards. You will need to select your drive and *erase* the partition. This will create a fresh partition and get you going onto the next step.

Installing the operating system

Once your SD card is prepped and ready to go, you can start to install the operating system. Depending on the OS you are looking at installing, the instructions may be slightly different but we will be covering installing Android and Raspbian/Bananian, which are the most popular choices for the Banana Pi M1 and M2 units.

Installing Android

Android is one of the main reasons people come looking for the Banana Pi. With the beefed up stats and the ability to run one of the most popular operating systems for media consumption, it makes perfect sense to look at this option for something like a media PC or something else along those lines. Currently, Android can only be set up for the Banana Pi using Windows. This is due to the fact that the SD card needs to be set up in a certain way to operate, which is slightly different than installing Linux. You cannot use the `dd` command like you can for installing operating systems. You will need to get the PhoenixCard software shown in the following screenshot:



disk H:\

D:\Software\BananaPi OS\Android_For_BananaPi_v2_0.img

Write Mode: ( ) Product [✓ Startup] ( ) Burn Key

```
Device OK, the size of the device is 15178 M.
```

This software is admittedly somewhat difficult to use. Some of the UI is not translated into English and it doesn't always work on the first try. Once the software is launched you will see the UI and you'll notice this:

- The first thing on the very top is the **DiskCheck** button. You will use this to check the disk you plan to write to, once it is selected from the drop-down.
- The **Img File** button is used to locate the `.img` file of the OS you downloaded from the Banana Pi website.
- It is important to change the write mode to startup so that the SD card is written so it can be booted from.
- You will then need to click on **Format to Normal**, which will start the process of making the SD card bootable.

Once this is complete a pop-up dialog will announce **Success!** and you will know it is good to burn the Android image. To do this you will need to:

1. Click on the **Burn** button, which will cause all kinds of output to spew into the small output window.
2. Once you see **magic complete** (in my opinion, the least descriptive thing ever but also my favorite output from any piece of software I have ever used), you are good to go.
3. When this is done, you can pop the SD card into your Banana Pi and plug it in. You will see a boot animation that may take longer than you are expecting to complete. This is normal and if you wait around for a minute or so you will see Android in all its glory running on your tiny computer.

You are able to install just about any `.apk` file you can think of. I have installed emulators, Netflix, XBMC, and more and they all seem to work. If you install the Android 4.2 version, you will have access to the Google Play Store, if you grabbed the Android 4.4 KitKat download you will be out of luck for now, but you can install Amazon Market to get going.

Installing Linux on your Banana Pi

Most projects aside from media PCs (and even some that are media PCs) will require you to install Linux on the board. The install process is slightly different depending on the OS that you have on your computer right now, similar to preparing your SD card. Regardless of the operating system on your computer, the end result will be the same: a full Linux installation that is ready for you to start hacking on.

Installing through Windows

Installing the OS through Windows is quick and painless. Downloading the extremely common utility Win32 Disk Imager will solve this for you. Now, follow these steps:

1. The interface is simple; you have an input field at the top where you can choose an image file from the **Image File** drop-down, you can choose the drive and buttons for starting the process here.

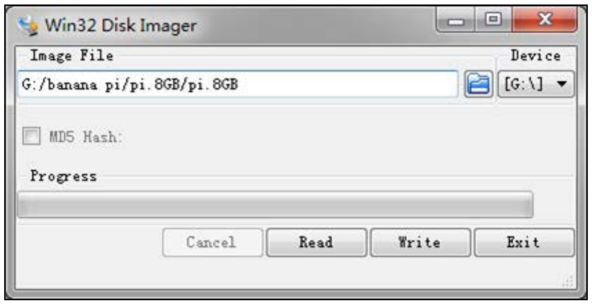


Image File: G:/banana pi/pi.8GB/pi.8GB

2. Once you have the drive and image file selected, it is just a matter of clicking on the **Write** button and waiting patiently. Once the write is complete, you are good to go. Insert the SD card into the Pi and boot it up.

Installing on the Mac and Linux

Both Mac OS users and Linux users can follow the same instructions. This process involves using the command line but doesn't require too much of you aside from knowing the drive info and having the image downloaded.

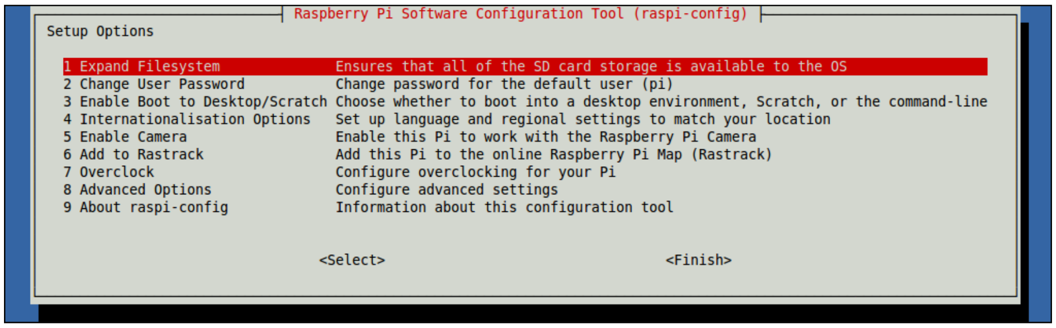
1. The first thing you are going to need to do is get a list of the drives on the system:
   ```
   sudo fdisk -l
   ```
2. Once you know which drive you are working with, you can check the hash key of the ZIP file to make sure it is the same as the download page. This step is optional but it will ensure you are installing the correct image.
   ```
   sha1sum [path]/[imagename]
   ```
3. You will need to unmount the drive so you can write to it.
   ```
   unmount  /dev/mmcblkxx
   ```
4. Writing the image to the drive is done using the `dd` command. This is a powerful command that is used to convert and copy files. The following command will write an `.img` file to a drive using 4M block sizes:
   ```
   sudo dd bs=4M if=[path]/[imgname] of=/dev/mmcblkxx
   ```
5. If the 4M block size does not work (it generally should) you can try again using 1M blocks. The smaller blocks will make the process take longer but it shouldn't take long to write.

Configuring your Linux installation

Now that we have a working installation, we can get to configuring it. If you installed Android it will be mostly configured out-of-the-box. If you installed Linux, you will need to set up your environment. The process is slightly different for each OS but I will be focusing on Raspbian and Bananian since they are the most common. The process for each of these operating systems is only slightly different.

Setting up Raspbian

The configuration process for Raspbian is identical to working with the Raspberry Pi since this operating system is a direct port from the Raspberry-flavored one. The configuration tool is brought up by running the `raspi-config` tool. This tool allows you to manage things such as resizing the filesystem and changing the host name.



```
Raspberry Pi Software Configuration Tool (raspi-config)
Setup Options

1 Expand Filesystem              Ensures that all of the SD card storage is available to the OS
2 Change User Password           Change password for the default user (pi)
3 Enable Boot to Desktop/Scratch Choose whether to boot into a desktop environment, Scratch, or the command-line
4 Internationalisation Options   Set up language and regional settings to match your location
5 Enable Camera                  Enable this Pi to work with the Raspberry Pi Camera
6 Add to Rastrack                Add this Pi to the online Raspberry Pi Map (Rastrack)
7 Overclock                      Configure overclocking for your Pi
8 Advanced Options               Configure advanced settings
9 About raspi-config             Information about this configuration tool

                  <Select>                      <Finish>
```

The `config` tool can be launched by running the following command (you will need to be root or run it with `sudo`):

```
sudo raspi-config
```

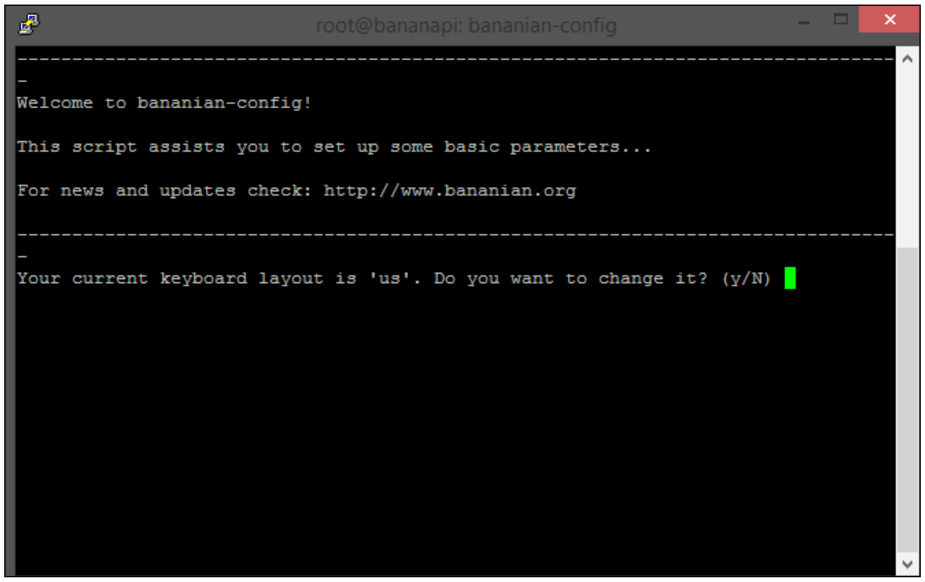
There are many different options in this menu and all of them are pretty useful. The following table will explain some of their uses. Not every project needs all of these configurations but they are good to have nonetheless.

Options	**Description**
Expand Filesystem	Ensures that all of the SD card storage is available to the OS
Change User Password	Change password for the default user (pi)
Enable Boot to Desktop/Scratch	Choose whether to boot into a desktop environment, Scratch, or the command line
Internationalization Options	Set up language and regional settings to match your location
Enable Camera	Enable this Pi to work with the Banana Pi Camera
Add to Rastrack	Add this Pi to the online Raspberry Pi Map (Rastrack)
Overclock	Configure overclocking for your Pi
Advanced Options	Configure advanced settings
About raspi-config	Information about this configuration tool

The menu can be navigated easily by using the arrow keys on your keyboard and pressing *Enter* to select a menu option. You can also press *Tab* to switch to **Finish** when you are done configuring your Pi.

Setting up Bananian

Bananian is intended to be super lightweight out-of-the-box. This allows it to work on basically all of the Banana Pi devices including some we are not covering in this book, such as the R1 router. Some of the main differences between Raspbian and Bananian are that Bananian is command line only out-of-the-box and uses the zsh command line as opposed to Bash. This won't affect us (and is actually my preference) while working with the Pi. The following screenshot shows you the Bananian command line:



```
--------------------------------------------------------------------------------
-
Welcome to bananian-config!

This script assists you to set up some basic parameters...

For news and updates check: http://www.bananian.org

--------------------------------------------------------------------------------
-
Your current keyboard layout is 'us'. Do you want to change it? (y/N)
```

Bananian has its own (less pretty) version of the `raspi-config` tool called `bananian-config`. This tool is less of a selection of tools and more of a wizard style tool, that will ask you step by step if you want to use each of its features.

The steps for the Bananian config tool are:

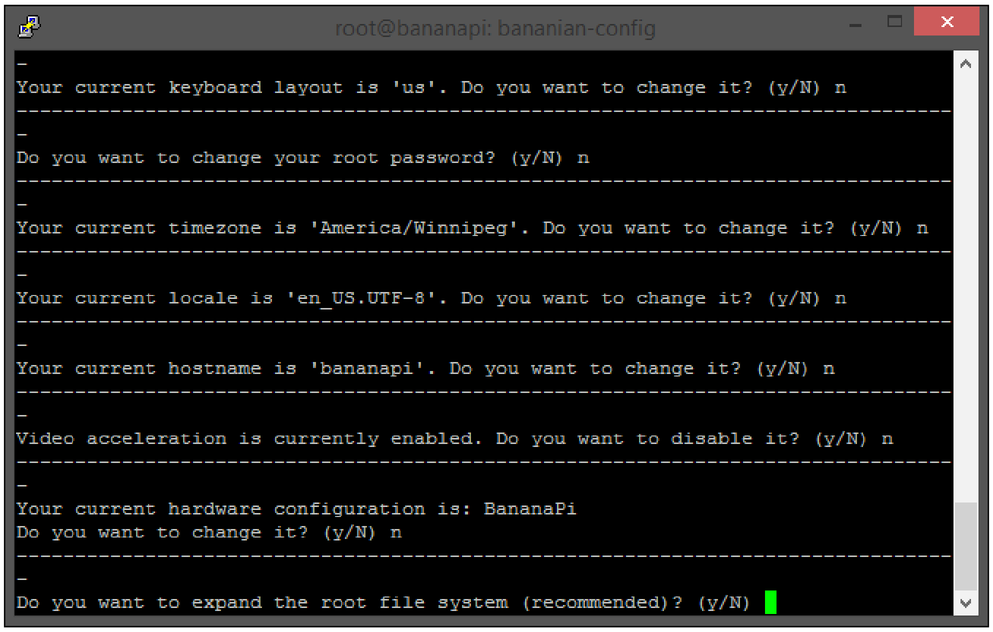
1. Changing the keyboard layout (the default value is `us`).
2. Changing the root password.
3. Changing the current time zone.

4. Changing the current locale.
5. Changing the hostname.
6. Enabling/disabling video acceleration.
7. Hardware configuration.
8. Expanding the root filesystem.

Overall, the Bananian and Raspbian config tools offer the same functionality but they do function differently. These tools are both important for setting up your system.

Expanding the root partition

This is one of the configuration options for both configuration tools. When you flash the OS onto the SD card, regardless of the size of the card, it will create specific partitions. If you were to use a 32 GB SD card you would be missing out on some space. This tool lets you expand the filesystem.



```
-
Your current keyboard layout is 'us'. Do you want to change it? (y/N) n
--------------------------------------------------------------------------------
-
Do you want to change your root password? (y/N) n
--------------------------------------------------------------------------------
-
Your current timezone is 'America/Winnipeg'. Do you want to change it? (y/N) n
--------------------------------------------------------------------------------
-
Your current locale is 'en_US.UTF-8'. Do you want to change it? (y/N) n
--------------------------------------------------------------------------------
-
Your current hostname is 'bananapi'. Do you want to change it? (y/N) n
--------------------------------------------------------------------------------
-
Video acceleration is currently enabled. Do you want to disable it? (y/N) n
--------------------------------------------------------------------------------
-
Your current hardware configuration is: BananaPi
Do you want to change it? (y/N) n
--------------------------------------------------------------------------------
-
Do you want to expand the root file system (recommended)? (y/N)
```

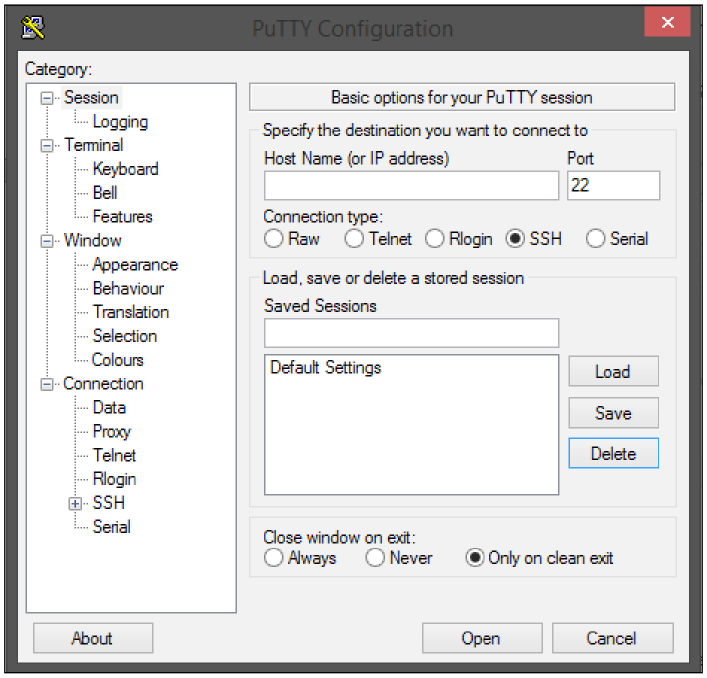
Once the Resize tool is used (it is fast), you will need to reboot the system, but it is worth it because now you are using your whole drive. If you had tried to install any software before doing this, you would have run out of room very quickly.

Connecting remotely

Not every project you will do will require you to use a display directly attached to the Banana Pi. You can connect to the device remotely from one of your other computers on the network and run the Pi *headless* (with no display). This is accomplished using SSH. By default, SSH is installed on Raspbian and Bananian out-of-the-box. For this, you will need to make sure that the Banana Pi is connected to the same network as your computer.

SSH from Windows

Windows does not come with an SSH client out-of-the-box. You need to download one. The most common application for this is called PuTTY. PuTTY is a free application you can download from its website (`http://www.chiark.greenend.org.uk/~sgtatham/putty/`). The application does quite a bit when you get right down to it but we just need the bare-bones configuration.



Specify the destination you want to connect to

Host Name (or IP address): Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Load, save or delete a stored session

Saved Sessions: Default Settings

Close window on exit: Always Never [✓ Only on clean exit]

1. To connect to a remote computer you will need its IP address. You can get this if you currently have a display connected to the Banana Pi by opening the terminal emulator and running the command:

```
ifconfig
```

2. This will produce a bunch of output that will include the information for each of your network interfaces. If you are wired; you will look for `eth0`, and if you are wireless; you will look for `wlan0`. The IP address you need will be preceded by `inet`. The output of the command looks like this:

```
eth0      Link encap:Ethernet  HWaddr 02:07:05:01:d2:44
          inet addr:192.168.1.6  Bcast:192.168.1.255
          Mask:255.255.255.0
          inet6 addr: fe80::7:5ff:fe01:d244/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:86523 errors:0 dropped:0 overruns:0
          frame:0
          TX packets:1649 errors:0 dropped:0 overruns:0
          carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9430022 (8.9 MiB)  TX bytes:115701 (112.9
          KiB)
          Interrupt:117 Base address:0xc000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

3. From this output you can see that the IP address we need is 192.168.1.6. We can enter this IP into the **Host Name** input field on PuTTY. When you connect, you will be prompted for the username and password and will be put into a command line session running on the Banana Pi.
4. If you need the IP address and do not have direct access to the Banana Pi you can generally get this information from your router, or there are various applications you can download to get this information from.

SSH from Linux or the Mac

For your benefit, if you are using a Mac computer or Linux you have a terminal emulator built in. Although you can download PuTTY on Linux, you don't need to, because all you need to do is run the SSH command. On either Mac or Linux open up your bash terminal and run this command:

Share this:

```
ssh user@ipaddress
```

The user is whichever user you plan to run as, and the IP address is specific to your device within your network. For example, if my IP address is 192.168.1.6 and I want to connect as root, I will run the command like so:

```
ssh root@192.168.1.6
```

Now I will be logged in as root. You will be prompted to remember the connection and be asked for the password. You can create an SSH key to automatically log you in, but for what we are doing this won't be necessary for now.

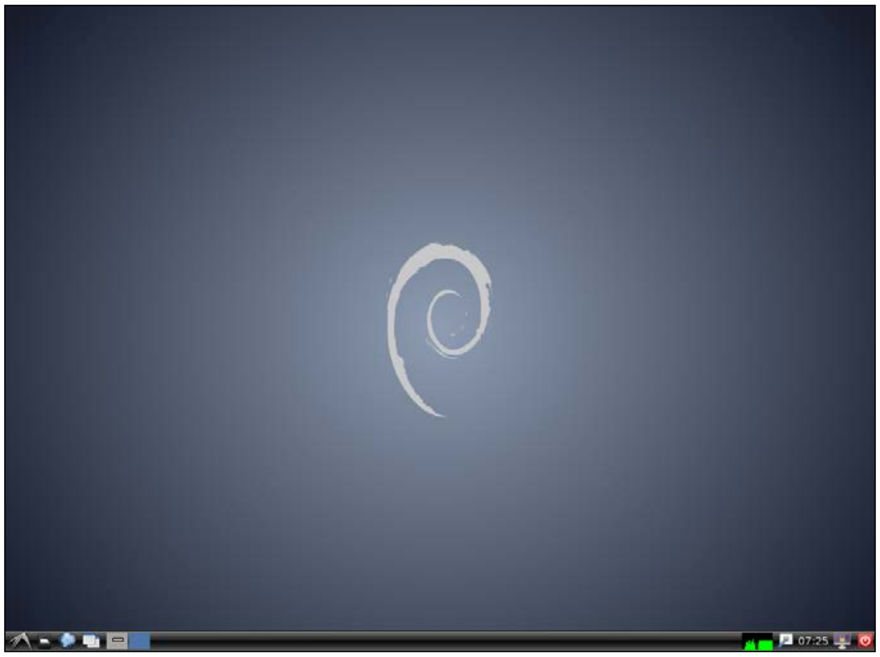
SSH remotely

It is possible to connect to your device over SSH from outside your network but this is something that is specific to your network. One thing that is common and that you will need to do, is forward the port you want to use for SSH (commonly 22) and route that connection to the Pi. This of course will open up a hole in your firewall and allow traffic over that port for that device. This means you will need to secure your connection in more ways such as by changing the default port, adding a new user, and restricting root access. This is useful for projects where you need to remotely connect such as file servers or remote web servers.

Depending on the applications you plan to run remotely you will need to open different ports. If you are accessing the Pi over SSH you will need port 22; if you are running a web server you will need 80, 8080, or whatever port you have decided on. By default, if you are using it as a remote database server for MySQL, you will need 3365. These ports are all configurable but you will need to take this into consideration depending on what you plan on running.

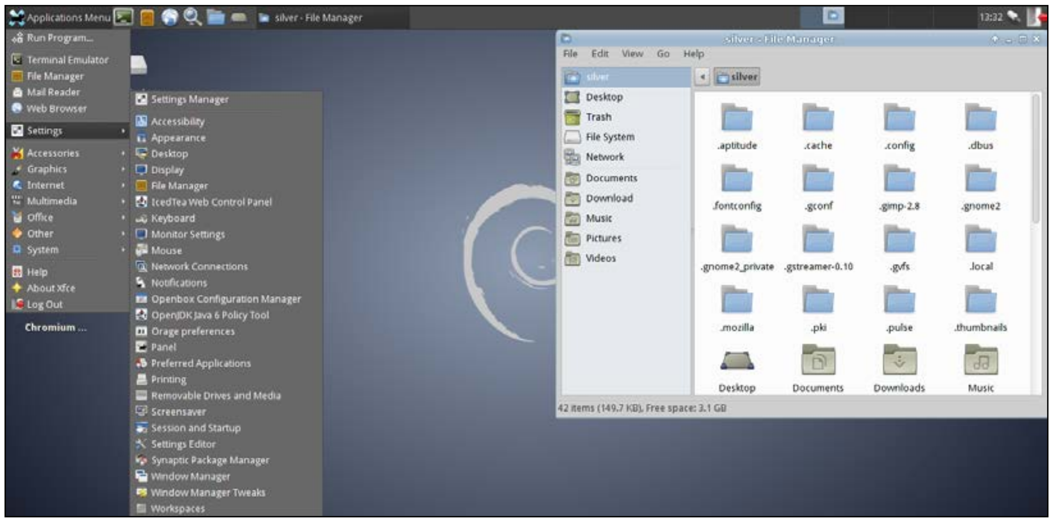
Setting up your desktop environment

Raspbian comes with a desktop environment out-of-the-box, Bananian does not. The nice thing about Linux is there are many different desktops to choose from. It is generally a good idea to stick to the lightweight options since the Pi's do not have a lot of graphic rendering power and are basically lightweight computers. The following screenshot shows the LXDE desktop, popular for lightweight distributions.



Raspbian comes with LXDE, a lightweight desktop environment (the clue is in the name). This is a pretty popular option because it is a classic layout, quick, and familiar to users who are not used to Linux because of its very Windows 7-inspired look. There are alternatives, though—for example, XFCE.

XFCE, as you can see from the following screenshot, is quite a bit different from LXDE but both are very customizable. XFCE will be more familiar to users who have used Linux before or even Mac users because of the use of the doc and menus on the top of the screen.



The desktop environment really comes down to personal preference because it really doesn't add anything or take anything away; it's just a question of aesthetics. You can even install both and switch between the two to try them out. To install LXDE on the Bananian you will need to run:

```
apt-get install task-lxde-desktop
```

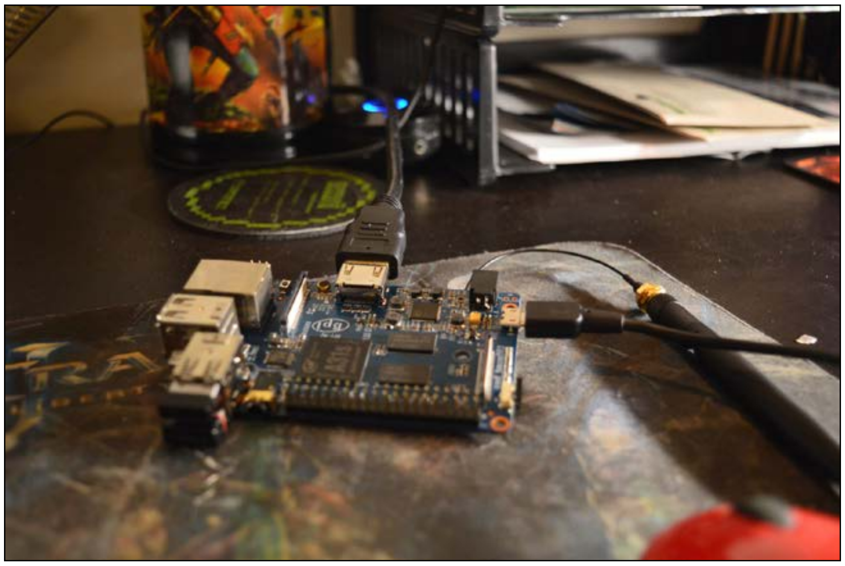
Once the installation is done, you will need to reboot but you will then be prompted with a login screen. From here you can open the application drawer in the bottom-left corner and launch a terminal to begin installing the software. To install XFCE on either operating system you will need to run the following command:

```
apt-get install xfce4-desktop
```

You will need to install the dependencies as well before this will work on Bananian because of the stripped down environment you have.

Summary

In this chapter, we covered a lot. You now should have an SD card prepared for the Banana Pi with your choice of operating system. You should also have a means of connecting to the Pi, either by SSH or with a display connected directly to it. You will also have a desktop environment you like that you will feel comfortable working in while working on projects. These are the basic skills you need to take your hardware from new to working. It doesn't take long but you now have a working credit-card-sized computer. Your Banana Pi likely looks like mine by now.



In the next chapter we are going to cover some more operating system-specific topics such as the filesystem on Linux, important commands that you will end up using frequently, how to install and find software you want to work with, interfacing with the GPIO pins from the command line, and interacting with hardware peripherals. By the end of the next chapter you will not only have a working computer, you will know how to use it effectively.

3
Linux and the Command Line

You might be feeling great now that you are up to speed with the hardware of the Banana Pi and have an operating system installed. Now, you are ready to use it. As not everybody is familiar with Linux, in this chapter, we will explore some of the ins and outs of Linux. We will also explore some important commands and the filesystem. Moreover, you will learn how to interface with some of the hardware on the Banana Pi.

This chapter will prepare you to work with the operating systems available for the Banana Pi, as most of them are Linux or Linux based. We will also explore using the fancy GPIO pins we talked about in the first chapter.

Linux is one of the most popular operating systems in the world. Whether you are aware of it or not, you are using Linux every day. Linux powers most of the home network routers, web servers, and even some of your electronics. If you have an Android phone, you are using Linux when you use your phone. Linux is not just for desktop computers. So, these skills may help you work with the technology around you, not just your Banana Pi.

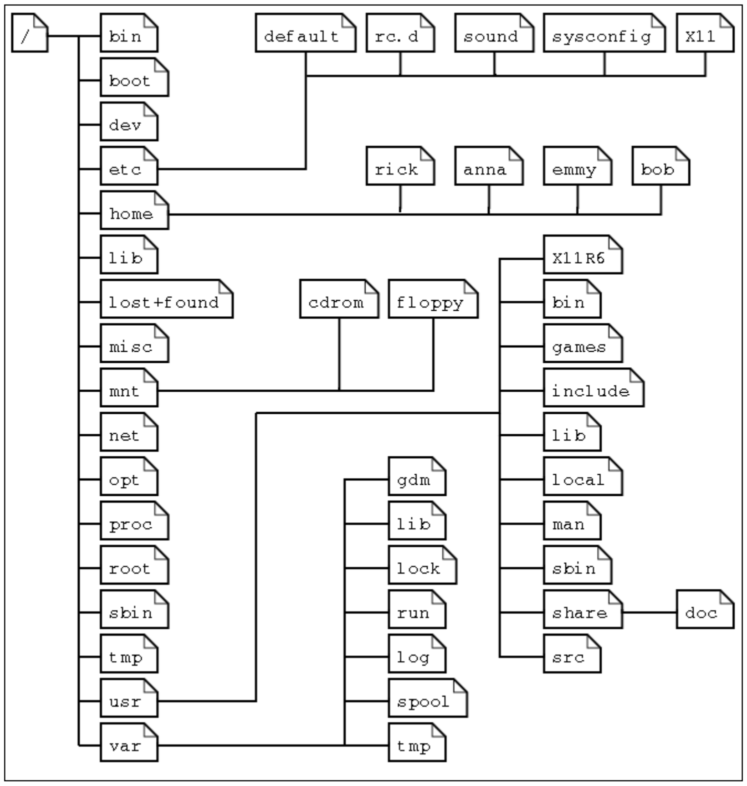
The filesystem

On the Linux filesystem (and Unix), everything is a file. If something is not a file, it is a process. There are specific files that are more than just files. Some examples of these files are pipes and sockets. There are several different styles of files, which are as follows:

- **Directories**: These files are lists of other files
- **Special files**: These are the mechanisms used for input and output; most of these files are located in the `/dev` directory

- **Links**: These are used to make a file visible in multiple locations
- **Sockets**: These are special files that provide interprocess networking; these sockets are protected by the filesystem's access control
- **Named pipes**: These form a way for processes to communicate with each other

Keep in mind that everything is a file in Linux. This is a little different from Windows, which uses files and folders to organize things. The following diagram shows a graphical representation of the Linux filesystem:

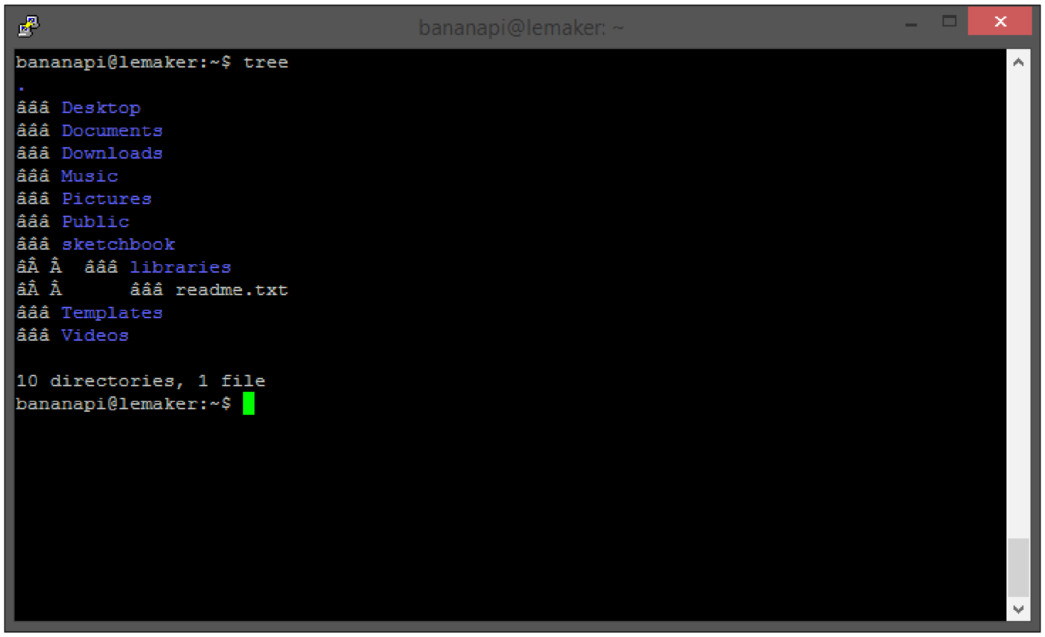


Everything on Linux stems from the root of the filesystem, which is represented by `/`. This is not the same as the root user, which is the super user of the system. The root of the filesystem contains several directories that are important for the operating system. The root of a Raspbian install looks like this:

```
drwxr-xr-x   2 root root 12288 Feb 13 09:24 bin
drwxr-xr-x   2 root root  4096 Jun  9  2014 boot
drwxr-xr-x  14 root root  4420 Jan  1  2010 dev
drwxr-xr-x 132 root root 12288 Feb 13 09:26 etc
drwxr-xr-x   3 root root  4096 Jun 25  2014 home
drwxr-xr-x  25 root root  4096 Feb 13 09:24 lib
drwx------   2 root root  4096 Jun  9  2014 lost+found
drwxr-xr-x   4 root root  4096 Jun  9  2014 media
drwxr-xr-x   2 root root  4096 Jan  6  2014 mnt
drwxr-xr-x   2 root root  4096 Feb  5  2014 opt
dr-xr-xr-x 149 root root     0 Jan  1  1970 proc
drwx------   5 root root  4096 Aug  6  2014 root
drwxr-xr-x  22 root root   740 Feb 13 09:00 run
drwxr-xr-x   2 root root 12288 Feb 13 09:24 sbin
drwxr-xr-x   2 root root  4096 Feb  5  2014 srv
dr-xr-xr-x  12 root root     0 Jan  1  2010 sys
drwxrwxrwt   5 root root  4096 Feb 13 09:18 tmp
drwxr-xr-x  11 root root  4096 Feb  9 10:04 usr
drwxr-xr-x  13 root root  4096 Feb  5  2014 var
```

At the root of the filesystem, you will find various directories that need to be included in our system. These directories hold files and folders for the boot loader, device files, and variable data.

The following screenshot shows a tree representation of the user's home directory:



```
bananapi@lemaker:~$ tree
.
âââ Desktop
âââ Documents
âââ Downloads
âââ Music
âââ Pictures
âââ Public
âââ sketchbook
âÂ Â   âââ libraries
âÂ Â       âââ readme.txt
âââ Templates
âââ Videos

10 directories, 1 file
bananapi@lemaker:~$
```

The following table describes the directories at the root of the Linux filesystem:

Directory	Description
`/bin`	Essential command binaries
`/boot`	Static files for the boot loader
`/dev`	Device files
`/etc`	Host-specific system configuration
`/lib`	Essential shared libraries and kernel modules
`/media`	Removable media mount point
`/mnt`	Temporary mount point for filesystems
`/opt`	Add-on application software packages
`/sbin`	Essential system binaries
`/srv`	Data for services provided by this system
`/tp`	Temporary files

Directory	Description
`/usr`	Secondary hierarchy
`/var`	Variable data

There are also several directories or symlinks that must be in the root directory but are considered optional. Although most Linux distributions use some of them, almost all of them will use the home directory. These are used to denote the different directories:

- `/`: The root directory
- `/home`: Users' home directories
- `/root`: The root user's home directory

The home directory is where you will be located when you first connect to the device over SSH. You can change this behavior. However, you will work out of this directory often, unless you are performing some specific tasks.

Inside your home directory, you will find the directories in which you will store your media, downloads, music, and so on. These directories will be the folders you work out of when using a **graphical user interface** (**GUI**). These directories include:

```
drwxr-xr-x 2 bananapi bananapi 4096 Jun  9  2014 Desktop
drwxr-xr-x 2 bananapi bananapi 4096 Jun  9  2014 Documents
drwxr-xr-x 2 bananapi bananapi 4096 Jun  9  2014 Downloads
drwxr-xr-x 2 bananapi bananapi 4096 Jun  9  2014 Music
drwxr-xr-x 2 bananapi bananapi 4096 Jun  9  2014 Pictures
drwxr-xr-x 2 bananapi bananapi 4096 Jun  9  2014 Public
drwxr-xr-x 2 bananapi bananapi 4096 Jun  9  2014 Templates
drwxr-xr-x 2 bananapi bananapi 4096 Jun  9  2014 Videos
```

These folders are straightforward, and you can always create your own folder for whatever you need. On a fresh install of Bananian or Raspbian, the `/root` or `/home` directories may be empty. The desktop folder appears on your desktop when you are viewing the GUI.

Important commands

When you connect remotely to your Banana Pi, you will be given a command-line interface. This interface will allow you to traverse the filesystem using some simple commands. It is important to know these commands because they will not only let you move around, but will also enable you to create directories, edit files, change the owners, and so on. Here are some of the important commands, along with short descriptions from their man pages:

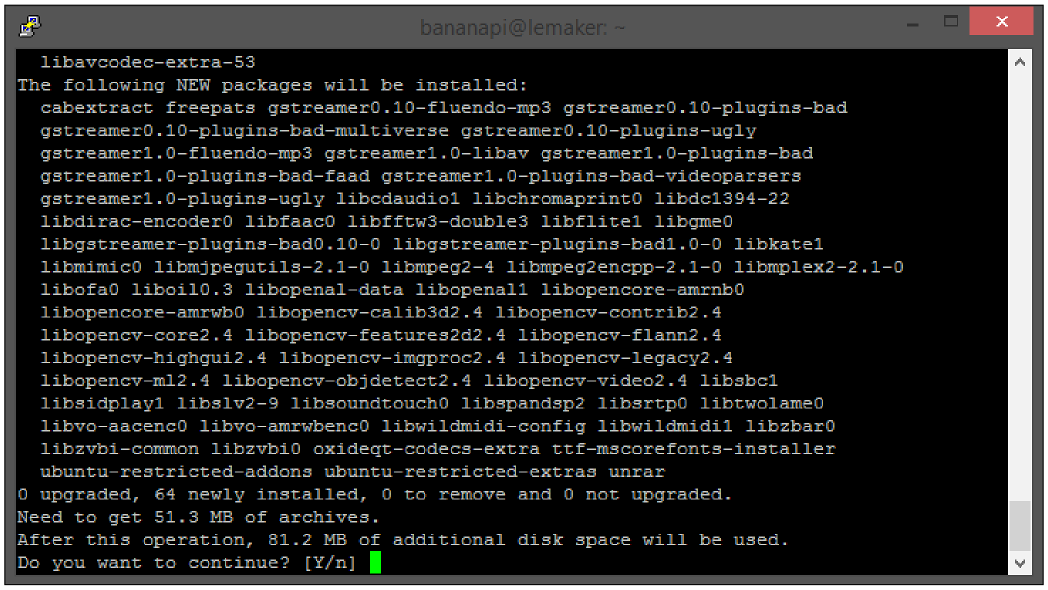
- `ls`: This lists information about the files (the current directory by default)
- `pwd`: This prints the full filename of the current working directory
- `mkdir`: This creates the directory (ies), if they do not already exist
- `chown`: This changes the user and/or group ownership of each given file
- `chmod`: This changes the file mode bits of each given file according to mode
- `touch`: This updates the access and modification times of each file to the current time
- `cat`: This concatenates file(s), or standard input, to standard output
- `cp`: This copies the source to destination, or multiple source(s) to a directory
- `mv`: This renames the source to destination, or moves source(s) to a directory
- `rm`: This removes each specified file; by default, it does not remove directories
- `ln`: This makes links between files

There are many more commands, but these are what you should memorize to start manipulating the filesystem.

Installing software

Arguably, one of the most important aspects of the operating system is its ability to install additional software. On Linux, software can come in many forms. You may install the package, download and install software from the Internet, or even download and compile software from the source. It is not difficult to install software on Linux. There are, of course, many more ways to do this than on an operating system such as Mac or Windows, where you generally install one way.

Since Raspbian is Debian based, the instructions for this section of the chapter will generally be Debian oriented. There are package managers other than apt, and there are, of course, package utilities other than `dpkg`. The command line is straightforward and the commands are easy to remember. The following screenshot shows the command line waiting for user entry to continue installing the ubuntu-restricted-extras on Lubuntu for the Banana Pi:



```
  libavcodec-extra-53
The following NEW packages will be installed:
  cabextract freepats gstreamer0.10-fluendo-mp3 gstreamer0.10-plugins-bad
  gstreamer0.10-plugins-bad-multiverse gstreamer0.10-plugins-ugly
  gstreamer1.0-fluendo-mp3 gstreamer1.0-libav gstreamer1.0-plugins-bad
  gstreamer1.0-plugins-bad-faad gstreamer1.0-plugins-bad-videoparsers
  gstreamer1.0-plugins-ugly libcdaudio1 libchromaprint0 libdc1394-22
  libdirac-encoder0 libfaac0 libfftw3-double3 libflite1 libgme0
  libgstreamer-plugins-bad0.10-0 libgstreamer-plugins-bad1.0-0 libkate1
  libmimic0 libmjpegutils-2.1-0 libmpeg2-4 libmpeg2encpp-2.1-0 libmplex2-2.1-0
  libofa0 liboil0.3 libopenal-data libopenal1 libopencore-amrnb0
  libopencore-amrwb0 libopencv-calib3d2.4 libopencv-contrib2.4
  libopencv-core2.4 libopencv-features2d2.4 libopencv-flann2.4
  libopencv-highgui2.4 libopencv-imgproc2.4 libopencv-legacy2.4
  libopencv-ml2.4 libopencv-objdetect2.4 libopencv-video2.4 libsbc1
  libsidplay1 libslv2-9 libsoundtouch0 libspandsp2 libsrtp0 libtwolame0
  libvo-aacenc0 libvo-amrwbenc0 libwildmidi-config libwildmidi1 libzbar0
  libzvbi-common libzvbi0 oxideqt-codecs-extra ttf-mscorefonts-installer
  ubuntu-restricted-addons ubuntu-restricted-extras unrar
0 upgraded, 64 newly installed, 0 to remove and 0 not upgraded.
Need to get 51.3 MB of archives.
After this operation, 81.2 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

The Apt package manager

In my opinion, Apt is one of the best package managers available. It is far superior to the homebrew manager for Mac and better than all package managers for Windows. Apt allows you to install software in the form of packages via the command line without the need to search the Internet for a download link. For example, if you want to install the VLC media player to view media on the Banana Pi, you could install it by running a simple command:

```
sudo apt-get install vlc
```

This command will install the software to a computer. The computer will then allow you to launch it through the application menu on the GUI or via the command line. Apt is also the tool you will run when updating and upgrading the software on your Pi. The Apt package manager allows you to:

- Install applications
- Remove applications
- Keep applications up to date

Apt is intended to be used to resolve dependency problems. It works well with `dpkg`, another tool used by Debian-based Linux distributions to handle the actual installation. If you are trying to install software through Apt, but it is not listed in your sources, you can add it using the following command:

```
sudo add-apt-repository ppa:<repo>
sudo apt-get update
sudo apt-get install <repo>
```

This command will effectively add the repo and install the software to the computer.

The dpkg tool

The `dpkg` tool is used to manage the packages being installed. Apt uses this tool to actually install and remove the packages you install through it. You can use `dpkg` manually to install files as if they were EXE files on Windows, for example. This tool is powerful and can be used to:

- Install packages
- Extract single files
- Remove packages
- Remove packages and their configuration files

If you download a `.deb` file, for example Skype from the Internet, you can use the `dpkg` tool to install it. If you downloaded `vlc.deb` to the `/home/user/Downloads/` directory, use this command to install it:

```
sudo dpkg -install ~/Downloads/vlc.deb
```

This command will act in a manner similar to the process of installing apt; it will install the package. The following icon shows the `.deb` icon visible on the GUI:

Interfacing with GPIO pins

As mentioned earlier, GPIO pins are a great way to interface with physical devices such as buttons and LEDs. Python is one of the most popular scripting languages on the Banana Pi (and Linux in general). When working with the Raspberry Pi, you have access to `RPi.GPIO`, a library to work with the GPIO pins in Python. Since the Banana Pi is slightly different, there is a modified version you will need to grab from the LeMaker GitHub page at `https://github.com/LeMaker/RPi.GPIO_BP`.

A single command will download the package from GitHub for you onto the Banana Pi:

```
git clone https://github.com/LeMaker/RPi.GPIO_BP -b bananapi
```

You can easily install this library by running the following commands:

```
sudo apt-get update
sudo apt-get install python-dev
cd ./RPi.GPIO_BP
python setup.py install
sudo python setup.py install
```

You will need to install `setup.py` both with and without `sudo`. This will facilitate the installation of the necessary components to the root user as well as your current user. Once this is done, you will be ready to start interfacing with the GPIO pins.

Interacting with the GPIO pins in Python

So, now that you have the Banana Pi flavor of the GPIO library installed, we can start to write some basic code to interface with the pins. We will start by creating the Python script we want to work out of, using the following command:

```
sudo nano gpio.py
```

This command will open the nano editor in which we can actually write some Python code:

```
#!/usr/bin/env python
import RPi.GPIO as GPIO

#Use BCM numbering
GPIO.setmode(GPIO.BCM)
```

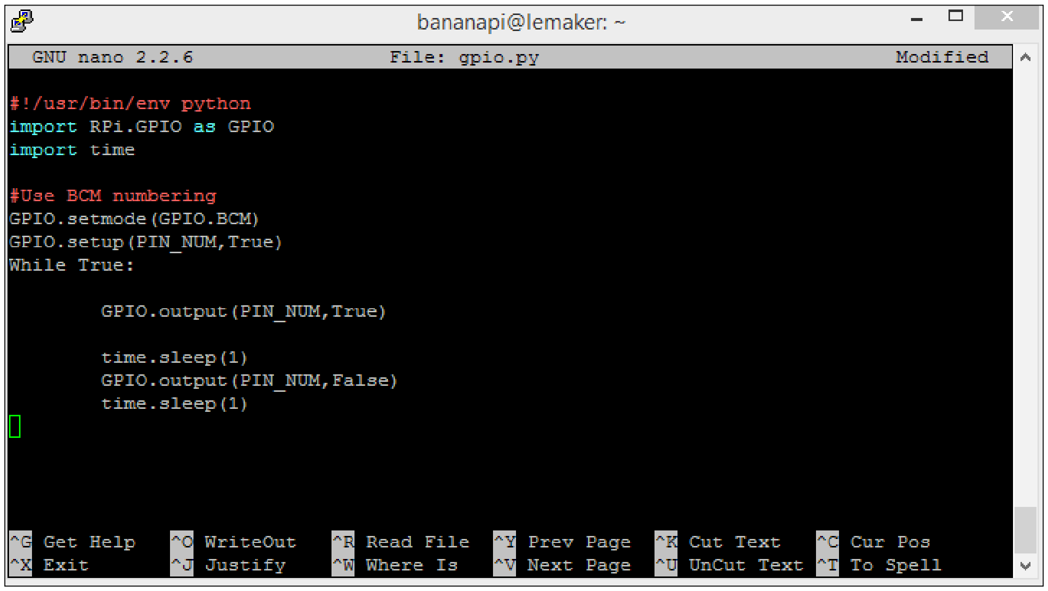
This command will tell the program to use the BCM option, which will use the Broadcom SoC numbering. This numbering is actually different from the versions of Raspberry Pi. However, we don't need to worry about that for the Banana Pi, because there is only one version currently.

We will take what we just wrote and add some script to it in order to create a simple script that will blink an LED on pin 7:

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import time
PIN_NUM = 7
#Use BCM numbering
GPIO.setmode(GPIO.BCM)
GPIO.setup(PIN_NUM, GPIO.OUT)
GPIO.output(PIN_NUM, GPIO.LOW)
while True:
        GPIO.output(PIN_NUM, GPIO.HIGH)
        time.sleep(1)
        GPIO.output(PIN_NUM, GPIO.LOW)
        time.sleep(1)
```

This simple script illustrates how easy it is to write a small amount of code and interact with a physical piece of hardware. We will work more with the GPIO pins in the upcoming sections while we tackle topics such as sensors.

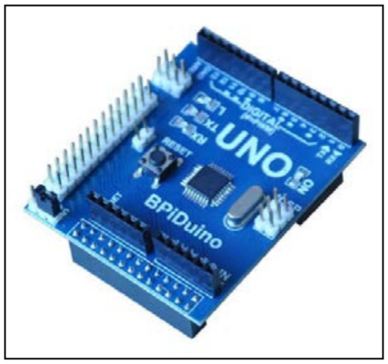
The following screenshot shows the code we just wrote in the nano editor on the Banana Pi:



```
bananapi@lemaker: ~
  GNU nano 2.2.6                File: gpio.py                          Modified

#!/usr/bin/env python
import RPi.GPIO as GPIO
import time

#Use BCM numbering
GPIO.setmode(GPIO.BCM)
GPIO.setup(PIN_NUM,True)
While True:

        GPIO.output(PIN_NUM,True)

        time.sleep(1)
        GPIO.output(PIN_NUM,False)
        time.sleep(1)


^G Get Help   ^O WriteOut   ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

Interacting with hardware

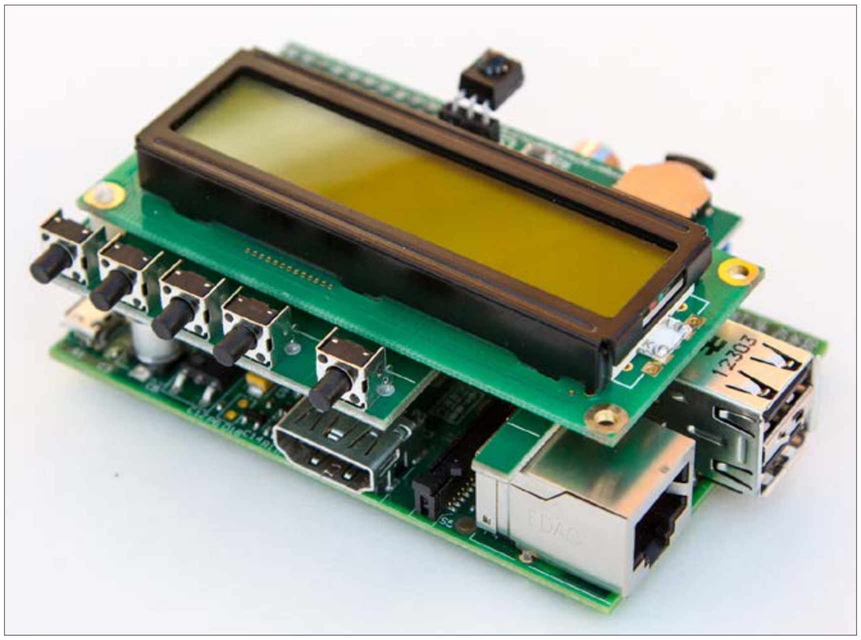
You may need to interact with all kinds of hardware on the Banana Pi. Whether it is a SATA drive or a peripheral, such as the PiFace command and control over the GPIO, there is a lot of hardware you can add to the Banana Pi. The following image shows the Banana Pi Uno board, an expansion board released by the makers of the Banana Pi to give it the pin out of the Arduino Uno:



This is one of many expansion boards released for the Banana Pi. These accessories offer different functionalities, such as an LED matrix to additional GPIO pins. In addition to all the Banana Pi accessories, most of the Raspberry Pi accessories are compatible as well, including the PiFace command and control.

Using expansion boards

One of my favorite expansion boards is the PiFace command and control board. It is actually intended for the Raspberry Pi, but it works fine for the Banana Pi. The board adds a 16 x 2 LCD Display, five buttons, a toggle, and an IR receiver (the Banana Pi already has one). The following image shows the PiFace:



Setting up the PiFace

The point of this section is to show how easy it is to actually set up a piece of Raspberry Pi hardware with our Banana Pi. The first thing we will do is install the CAD software and reboot the system:

```
sudo apt-get install python3-pifacecad
sudo reboot
```

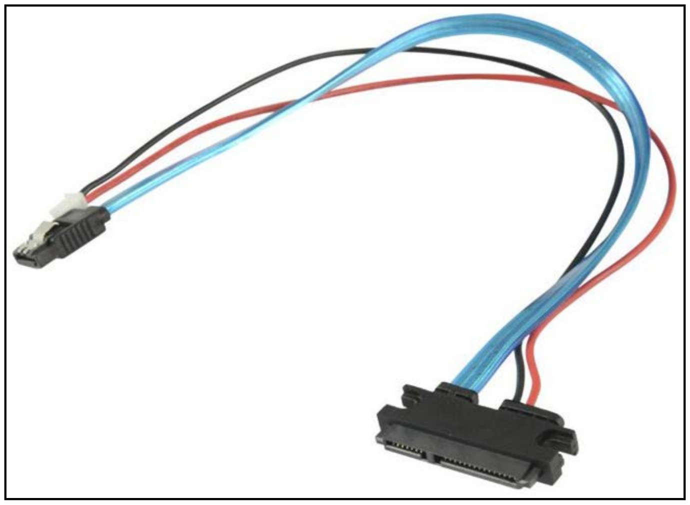
After the Pi has rebooted, we will run a script that will display some system information on the LCD display. This will include the IP address and temps for the Pi:

```
cd /usr/share/doc/python3-pifacecad/examples
python3 sysinfo.py
```

You can press *Ctrl* + *C* to exit the program. The following image shows the Banana Pi beside the Raspberry Pi model B. They are similar in size and layout. Thus, you can use the accessories freely between the two boards.

Mounting external drives

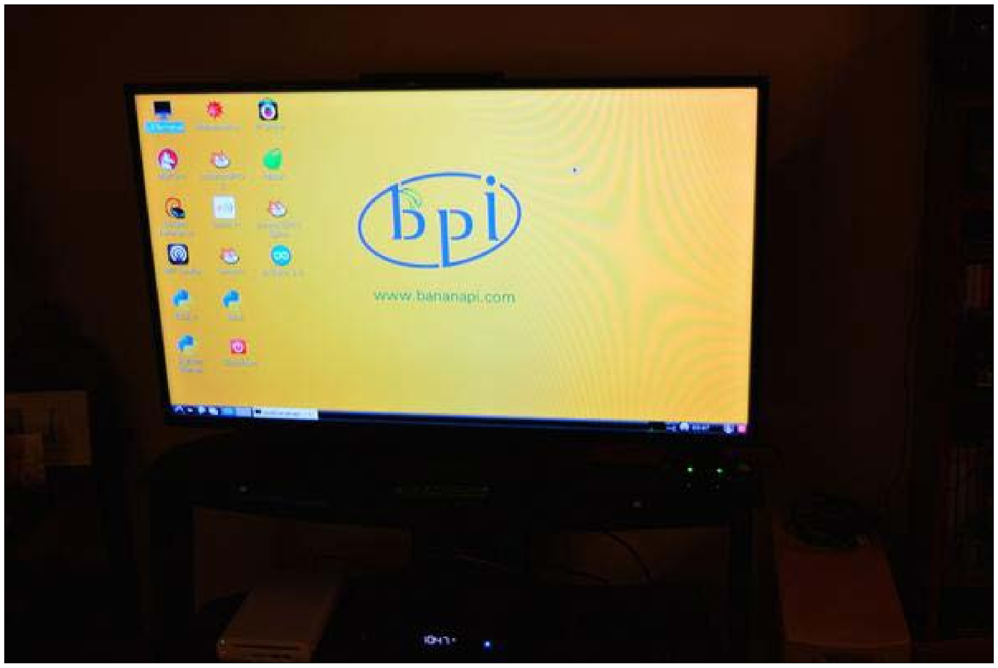
Since the Banana Pi has a SATA port, one of the first things people want to try out is mounting a drive. Unfortunately, you cannot boot directly off SATA. You need a microSD card in order to boot the device (although booting directly would have been the best). The following image shows the connector used to attach a 2.5-inch SATA drive to the Banana Pi:



If you connect the SATA drive to the Banana Pi with power and it is already formatted, you will see the drive mount and appear on your desktop as a mounted volume. It is important that you use the cable from the Banana Pi. Some other cables have the positive and negative lines reversed. This should work with any filesystem you are using on the drive. The `.ext4` filesystem seems to be the most stable one. Overall, I found the process of mounting drives to the Banana Pi fairly painless, if you are just looking to use it as storage space.

Summary

In this chapter, we touched on many topics relevant to getting comfortable with Linux. You learned how to use the hardware in a Linux environment and even got the GPIO pins working using Python and Linux. At this point, you have successfully got your OS installed and running, navigated the filesystem, and written some code to work with the GPIO pins. You are also able to connect a SATA drive to use as additional storage.



These skills are not confined only to the Banana Pi. They are good general technical skills that you should have. The Linux filesystem is similar to what you will find on a Unix system, including Mac, which is based on BSD (Unix). Working with the GPIO pins directly translates to other single-board computers that use GPIO based on the Raspberry Pis.

In the following chapter, you will learn more about programming on the Pi. You got a taste for it in this chapter, but in the next chapter, we will explore additional programming languages that you can use on the Banana Pi. You will also learn about different tools you can use to write the code that will power your next project.

We will explore languages such as Ruby, Bash, Python, and Go in order to write clean and efficient code that will interact with hardware components such as LEDs, switches, and other networked Pis.

4
Programming on the Pi

In the previous chapter, you became familiar with Linux and its filesystem as well as the command-line interface and some commands that you needed to master to use the system effectively. In this chapter, you are going to learn about some editors and the programming languages that are available on the Pi and Linux. These tools will help you write the code that will interact with the hardware through GPIO and on the Pi as a server.

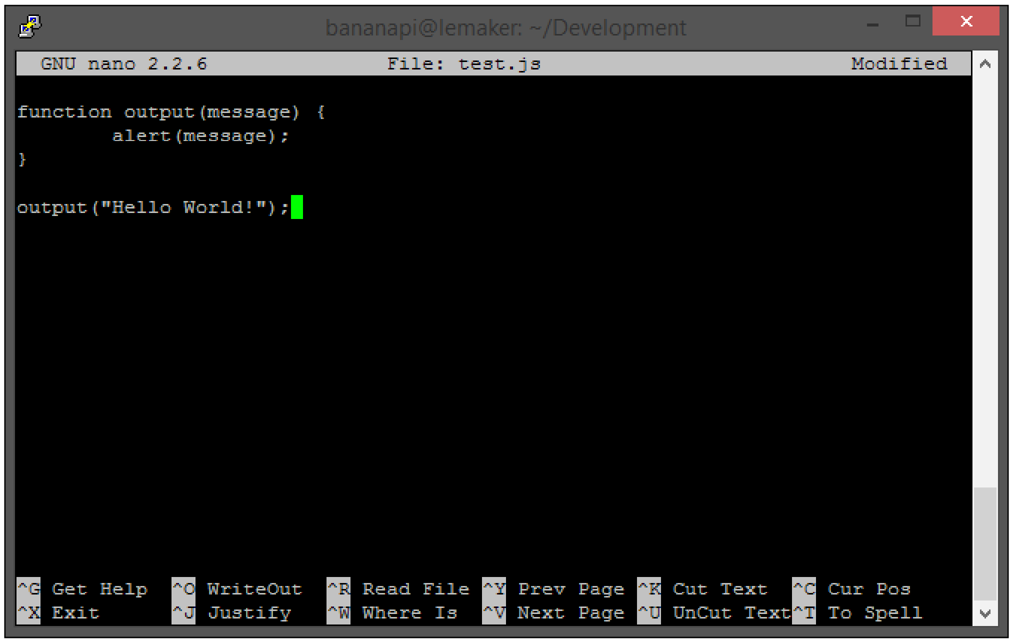
Coding may be something that is new to you, or it may be something that you are an expert in. In this chapter, we will cover some fairly basic programming, as the focus is on the different languages that are available to you and the tools to use them. We will concentrate on some languages such as Ruby and Python that are not only easy for beginners but also popular with seasoned developers.

Different languages are good for different things. If you are going to write a web application, you might use Ruby on Rails. If you want to write a script to be run in the terminal, you may use Bash. If you wish to write a server-side program to manage some data, you may use Python. The beauty of Linux is that most of the packages and programming languages that you want to use are either installed by default or extremely easy to install.

Choosing your editor

There are many different integrated development environments (generally abbreviated as IDEs) to choose from on Linux. When working on the Banana Pi, you're limited to the software that will run on an ARM-based CPU. Hence, options such as Sublime Text are not available. Some options that you may be familiar with are available for general purpose code editing. Some tools are available for the command line, while others are GUI tools.

So, depending on whether you have a monitor or not, you will want to choose an appropriate tool. The following screenshot shows some JavaScript being edited via Nano on the command line:


bananapi@lemaker: ~/Development

```
  GNU nano 2.2.6              File: test.js                       Modified

function output(message) {
        alert(message);
}

output("Hello World!");
```

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit   ^J Justify   ^W Where Is ^V Next Page ^U UnCut Text^T To Spell

Command-line editors

The command line is a powerful tool. If you master it, you will rarely need to leave it. There are several editors available for the command line. There has been an ongoing war between the users of two editors: GNU Emacs and Vim. There are many editors like Nano (which is my preference), but the war tends to be between the two aforementioned editors.

The Emacs editor

This is my least favorite command-line editor (just my preference). Emacs is a GNU-flavored editor for the command line. It is often installed by default, but you can easily install it if it is missing by running a quick command, as follows:

```
sudo apt-get install emacs
```

Now, you can edit a file via the CLI by using the following code:

```
emacs <command-line arguments> <your file>
```

The preceding code will open the file in Emacs for you to edit. You can also use this to create new files. You can save and close the editor with a couple of key combinations:

- *Ctrl* + *X* and *Ctrl* + *S*
- *Ctrl* + *X* and *Ctrl* + *C*

Thus, your document will be saved and closed.

The Vim editor

Vim is actually an extension of Vi, and it is functionally the same thing. Vim is a fine editor. Many won't personally go out of their way to not use it. However, people do find it a bit difficult to remember all the commands. If you do get good at it, though, you can code very quickly. You can install Vim with the command line:

```
sudo apt-get install vim
```

Also, there is a GUI version available that allows interaction with the mouse; this is functionally the same program as the Vim command line. You don't have to be confined to the terminal window. You can install it with an identical command:

```
sudo apt-get install vim-gnome
```

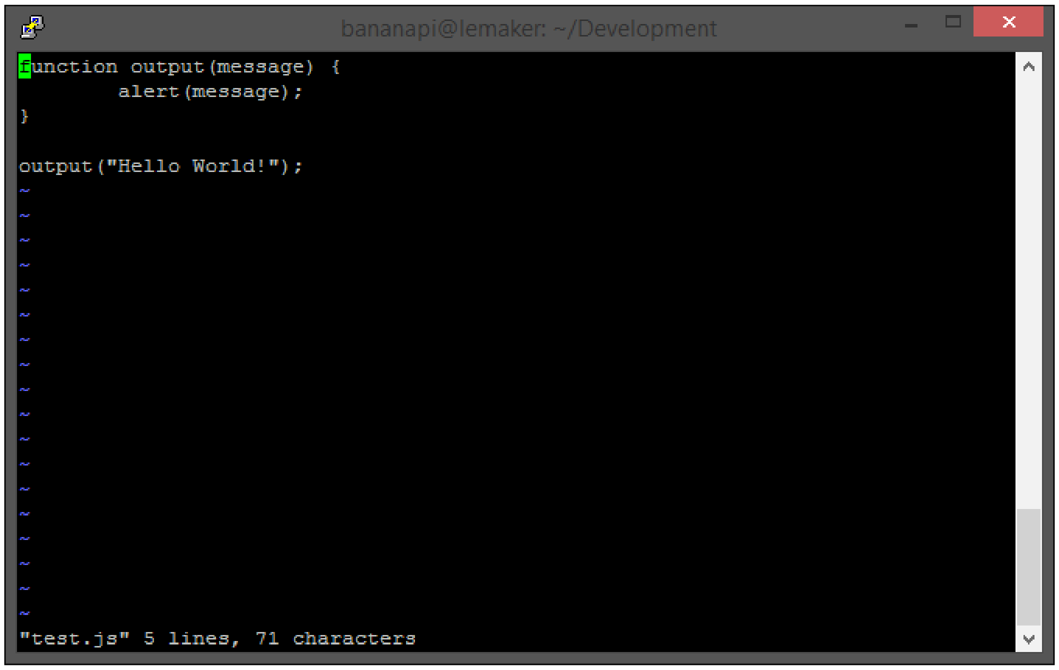
You can edit files easily with Vim via the command line for both Vim and Vim-Gnome, as follows:

```
vim <your file>
gvim <your file>
```

The gnome version will open the file in a window There is a handy tutorial that you can use to learn the commands of Vim. You can run the tutorial with the help of the following command:

```
vimtutor
```

This tutorial will teach you how to run this editor, which is awesome because the commands can be a bit complicated at first. The following screenshot shows Vim editing the file that we used earlier:



```
function output(message) {
        alert(message);
}

output("Hello World!");
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"test.js" 5 lines, 71 characters
```

The nano editor

The nano editor is my favorite editor for the command line. This is probably because it was the first editor that I was exposed to when I started to learn Linux and experiment with the servers and eventually, the Raspberry Pi and Banana Pi. The nano editor is generally considered the easiest to use and is installed by default on the Banana Pi images. If, for some reason, you need to install it, you can get it quickly with the help of the following command:

```
sudo apt-get install nano
```

The editor is easy to use. It comes with several commands that you will use frequently. To save and close the editor, use the following key combinations:

- *Ctrl* + *O*
- *Ctrl* + *X*

You can get help at any time by pressing *Ctrl* + *G*.

Graphic editors

With the exception of gVim, all the editors we just talked about live on the command line. If you are more accustomed to graphical tools, you may be more comfortable with a full-featured IDE. There are a couple of choices in this regard that you may be familiar with.

These tools are a little heavier than the command-line tools because you will need to not only run the software, but also render the window. This is not as much of a big deal on the Banana Pi as it is on the Raspberry Pi, because we have more RAM to play with. However, if you have a lot of programs running already, it might cause some performance issues.

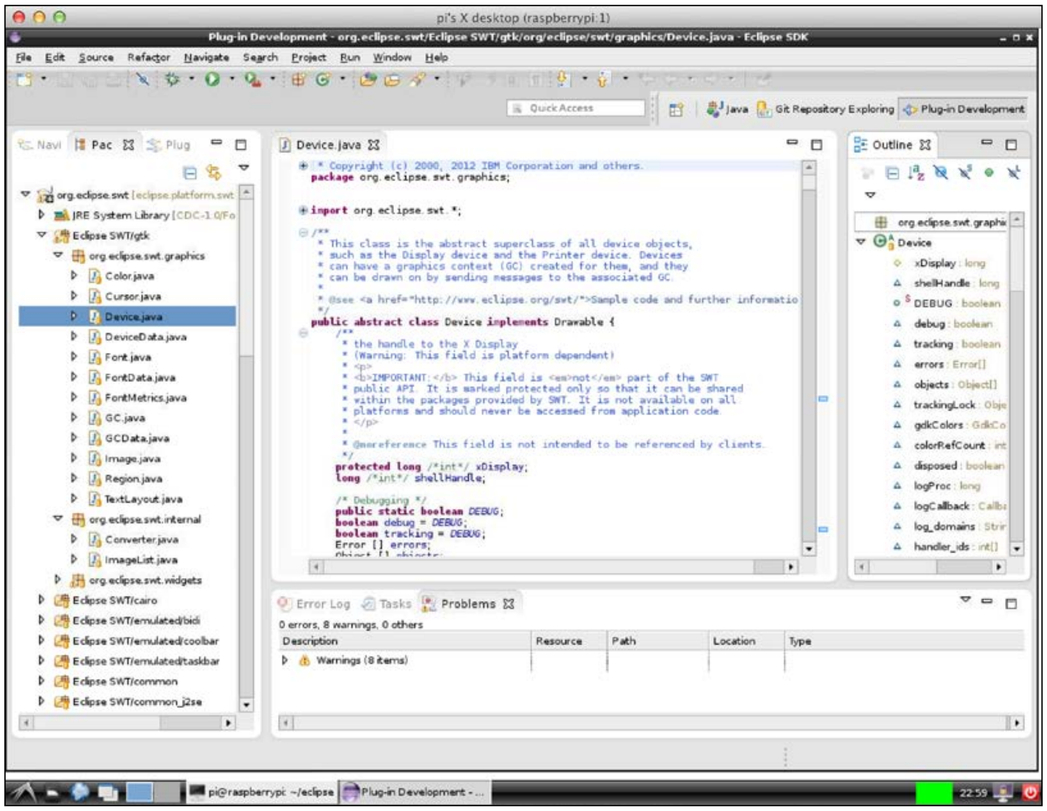
Eclipse

Eclipse is a very popular IDE that is available for everything. You can use it to develop all kinds of systems and use all kinds of programming languages. This is a tool that can be used to do professional development. There are a lot of plugins available in this IDE. It is also used to develop apps for Android (although Android Studio is also available now).

Eclipse is written in Java. Hence, in order to make it work, you will require a Java Runtime Environment. The Banana Pi should come equipped with the Java development and runtime environments. If this is not the case, they are not difficult to install. In order to grab the proper version of Eclipse and avoid browsing all the specific versions on the website, you can just install it via the command line by entering the following code:

```
sudo apt-get install eclipse
```

Once Eclipse is installed, you will find it in the application menu under programming tools. The following screenshot shows the Eclipse IDE running on the Banana Pi:


pi's X desktop (raspberrypi:1)
Plug-in Development - org.eclipse.swt/Eclipse SWT/gtk/org/eclipse/swt/graphics/Device.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Git Repository Exploring
Plug-in Development
Device.java
Outline
Error Log
Tasks
Problems

```
 * Copyright (c) 2000, 2012 IBM Corporation and others.
package org.eclipse.swt.graphics;

import org.eclipse.swt.*;

/**
 * This class is the abstract superclass of all device objects,
 * such as the Display device and the Printer device. Devices
 * can have a graphics context (GC) created for them, and they
 * can be drawn on by sending messages to the associated GC.
 *
 * @see <a href="http://www.eclipse.org/swt/">Sample code and further informatio
 */
public abstract class Device implements Drawable {
    /**
     * the handle to the X Display
     * (Warning: This field is platform dependent)
     * <p>
     * <b>IMPORTANT:</b> This field is <em>not</em> part of the SWT
     * public API. It is marked protected only so that it can be shared
     * within the packages provided by SWT. It is not available on all
     * platforms and should never be accessed from application code.
     * </p>
     *
     * @noreference This field is not intended to be referenced by clients.
     */
    protected long /*int*/ xDisplay;
    long /*int*/ shellHandle;

    /* Debugging */
    public static boolean DEBUG;
    boolean debug = DEBUG;
    boolean tracking = DEBUG;
    Error [] errors;
```

0 errors, 8 warnings, 0 others

Description	Resource	Path	Location	Type
Warnings (8 items)				

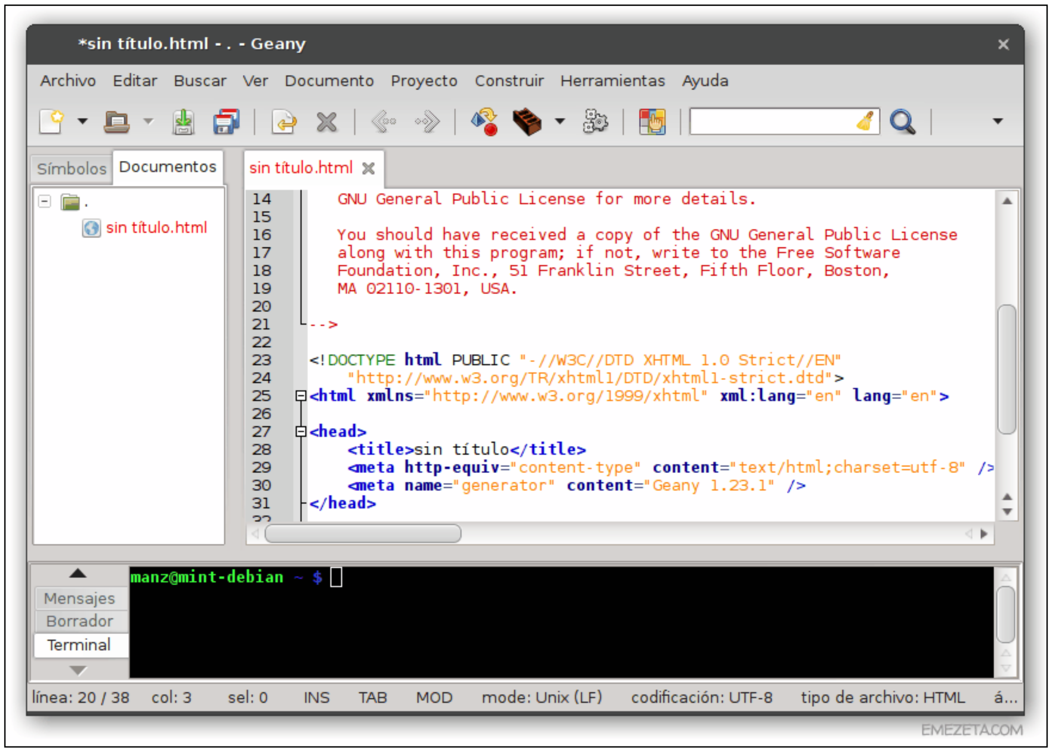
The Geany IDE

Geany is a lighter weight IDE than Eclipse although the former is not quite fully featured. It is a clean UI that can be customized and used to write a lot of different programming languages. Geany was one of the first IDEs I ever used when first exploring Linux when I was a kid.

Geany does not come preinstalled on the Banana Pi images, but it is easy to get via the command line:

```
sudo apt-get install geany
```

Depending on what you plan to do code-wise on the Banana Pi, Geany may be your best bet. It is GUI-based and offers quite a bit of functionality. However, it is a lot faster to load than Eclipse. It may seem familiar for Windows users, and they might find it easier to operate since it resembles Windows software. The following screenshot shows Geany on Linux:


*sin título.html - . - Geany
Archivo Editar Buscar Ver Documento Proyecto Construir Herramientas Ayuda
Símbolos Documentos
sin título.html
Mensajes
Borrador
Terminal
EMEZETA.COM

```
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
    MA 02110-1301, USA.

-->

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
    <title>sin título</title>
    <meta http-equiv="content-type" content="text/html;charset=utf-8" />
    <meta name="generator" content="Geany 1.23.1" />
</head>
```

```
manz@mint-debian ~ $
```

línea: 20 / 38 col: 3 sel: 0 INS TAB MOD mode: Unix (LF) codificación: UTF-8 tipo de archivo: HTML á...

Both of these editors, Geany and Eclipse, are not specific to a particular programming language, but they both are slightly better for certain languages. Geany tends to be better for web languages such as HTML, PHP, JavaScript, and CSS, while Eclipse tends to be better for compiled languages such as C++, Go, and Java as well as PHP and Ruby with plugins.

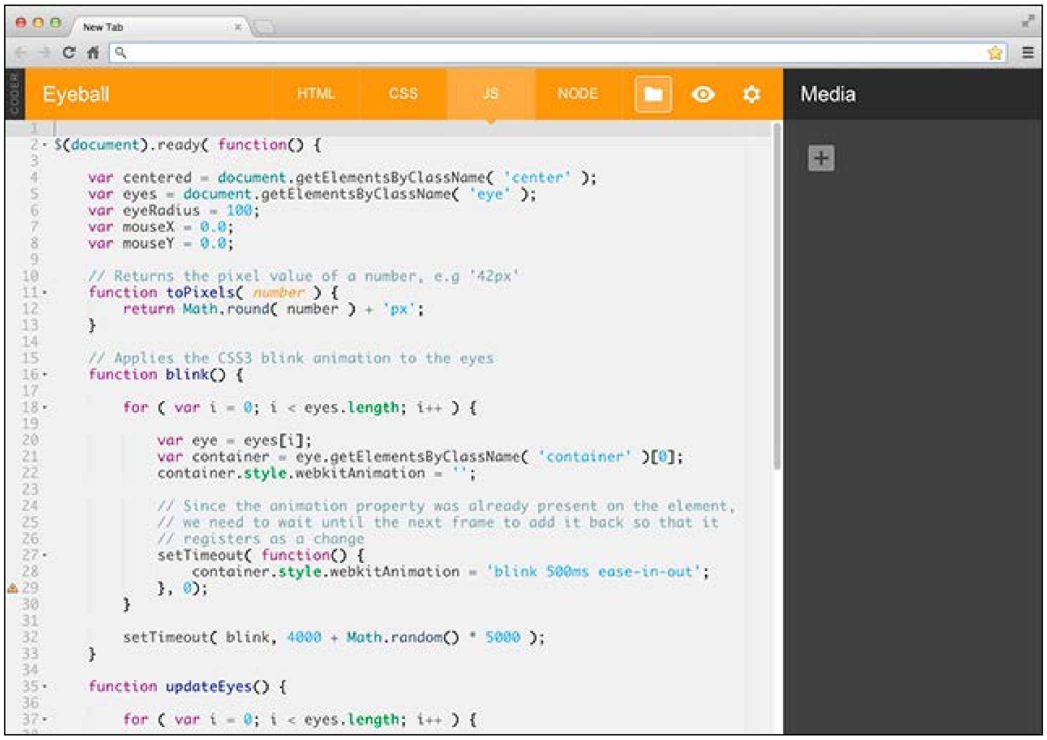
If you plan to write scripts or languages that are intended to be run from the command line such as Bash, Ruby, or Python, you may want to stick to the command line and use an editor such as Vim or Nano. It is worth your time to play around with the editors and find your preferences.

Web IDEs

In addition to the command line and GUI editors, there are a couple of web-based IDEs. These essentially turn your Pi into a code server, which allows you to run and even execute certain types of code on an IDE written in web languages. These IDEs are great for learning code, but they are not really replacements for the solutions that were listed previously.

Google Coder

Google Coder is an educational web IDE that was released as an open source project by Google for the Raspberry Pi. Although there is a readily available image for the Raspberry Pi, we can manually install it for the Banana Pi. The following screenshot shows the Google Coder's interface:



```

$(document).ready( function() {

    var centered = document.getElementsByClassName( 'center' );
    var eyes = document.getElementsByClassName( 'eye' );
    var eyeRadius = 100;
    var mouseX = 0.0;
    var mouseY = 0.0;

    // Returns the pixel value of a number, e.g '42px'
    function toPixels( number ) {
        return Math.round( number ) + 'px';
    }

    // Applies the CSS3 blink animation to the eyes
    function blink() {

        for ( var i = 0; i < eyes.length; i++ ) {

            var eye = eyes[i];
            var container = eye.getElementsByClassName( 'container' )[0];
            container.style.webkitAnimation = '';

            // Since the animation property was already present on the element,
            // we need to wait until the next frame to add it back so that it
            // registers as a change
            setTimeout( function() {
                container.style.webkitAnimation = 'blink 500ms ease-in-out';
            }, 0);
        }

        setTimeout( blink, 4000 + Math.random() * 5000 );
    }

    function updateEyes() {

        for ( var i = 0; i < eyes.length; i++ ) {
```

The setup is fairly straightforward. We will clone the Git repo and install it with Node.js. If you don't have Git and Node.js installed, you can install them with a quick command in the terminal, as follows:

```
sudo apt-get install nodejs npm git
```

Once it is installed, we can clone the coder repo by using the following code:

```
git clone https://github.com/googlecreativelab/coder
```

After it is cloned, we will move into the directory and install it with the help of the following code:

```
cd ~/coder/coder-base/
npm install
```

It may take several minutes to install, even on the Banana Pi. Next, we will edit the `config.js` file, which will be used to configure the ports and IP addresses.

```
nano config.js
```

The preceding code will reveal the contents of the file. Change the top values to match the following:

```
exports.listenIP = '127.0.0.1';
exports.listenPort = '8081';
exports.httpListenPort = '8080';
exports.cacheApps = true;
exports.httpVisiblePort = '8080';
exports.httpsVisiblePort = '8081';
```

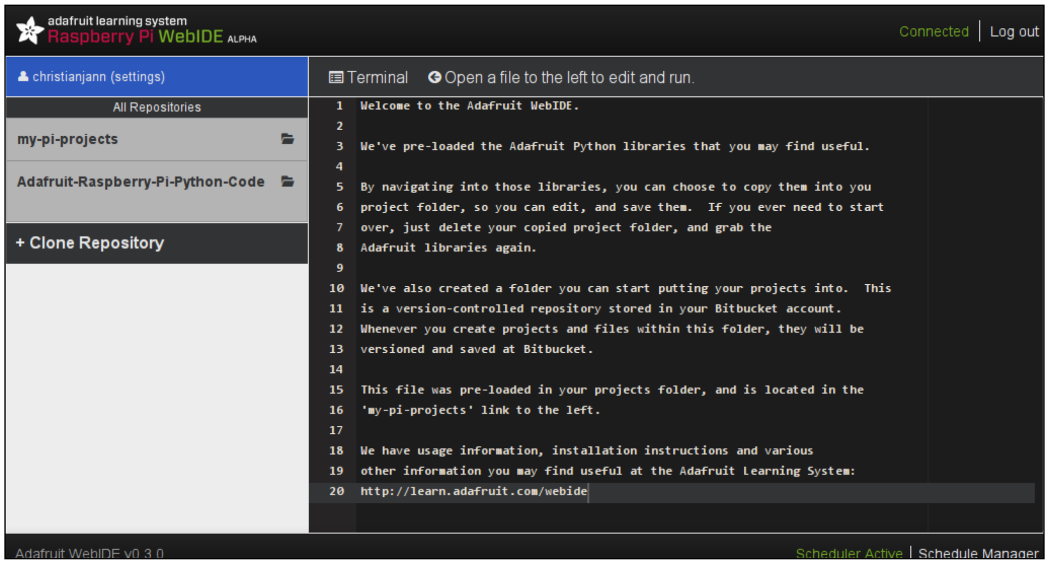
After you change the settings you need, run a server by using Node.js:

```
nodejs server.js
```

You should now be able to connect to the Pi in a browser either on it or on another computer and use Coder. Coder is an educational tool with a lot of different built-in tutorials. You can use Coder to learn JavaScript, CSS, HTML, and jQuery.

Adafruit WebIDE

Adafruit has developed its own Web IDE, which is designed to run on the Raspberry Pi and BeagleBone. Since we are using the Banana Pi, it will only run better. This IDE is designed to work with Ruby, Python, and JavaScript, to name a few. It includes a terminal via which you can send commands to the Pi from the browser. It is an interesting tool if you wish to learn how to code. The following screenshot shows the interface of the WebIDE:



```
Welcome to the Adafruit WebIDE.

We've pre-loaded the Adafruit Python libraries that you may find useful.

By navigating into those libraries, you can choose to copy them into you
project folder, so you can edit, and save them.  If you ever need to start
over, just delete your copied project folder, and grab the
Adafruit libraries again.

We've also created a folder you can start putting your projects into.  This
is a version-controlled repository stored in your Bitbucket account.
Whenever you create projects and files within this folder, they will be
versioned and saved at Bitbucket.

This file was pre-loaded in your projects folder, and is located in the
'my-pi-projects' link to the left.

We have usage information, installation instructions and various
other information you may find useful at the Adafruit Learning System:
http://learn.adafruit.com/webide
```

The installation of WebIDE is very simple compared to that of Google Coder, which took several steps. We will just run one command:

```
curl https://raw.githubusercontent.com/adafruit/Adafruit-WebIDE/alpha/
scripts/install.sh | sudo sh
```

After a few minutes, you will see an output that indicates that the server is starting. You will be able to access the IDE just like Google Coder—through a browser from another computer or from itself. It should be noted that you will be required to create a free Bit Bucket account to use this software.

Writing code on the Pi

By now, you are either playing with the IDEs, or you are ready to move on because you already have preferences. Either way, we want to write some code. There are different purposes behind writing different kinds of code. We will focus on Bash, Python, Ruby, and Go. These languages will give you a good range of general-purpose tools for different tasks on your Pi.

Bash scripting

Bash scripting is the purest form of coding on Linux. Bash scripts are essentially command-line commands strung together in a single file that can be executed. You can write scripts that run at the startup to run updates and pull some data onto the Pi. Shell scripts have the file extension `sh`.

The first line of a script should start with a shebang, which is represented by `#!`. This character sequence will tell the interpreter to look for the correct software to run the script that you have written. We will do this in bash so that we don't have to call programs with the bash command. Instead, we will run them directly. This is also useful for other languages, such as Ruby and Python.

You can write bash scripts by using whichever editor you like, but you will need to have the following specific line at the top of the script:

```
#!/bin/bash
```

I like to use the following script as a basic tool to control the GPIO pins on the Pi:

```
#!/bin/bash

function gpio()
{
    local verb=$1
    local pin=$2
    local value=$3

    local pins=($GPIO_PINS)
    if [[ "$pin" -lt ${#pins[@]} ]]; then
        local pin=${pins[$pin]}
    fi
```

```
        local gpio_path=/sys/class/gpio
        local pin_path=$gpio_path/gpio$pin

        case $verb in
            read)
                cat $pin_path/value
            ;;

            write)
                echo $value > $pin_path/value
            ;;

            mode)
                if [ ! -e $pin_path ]; then
                    echo $pin > $gpio_path/export
                fi
                echo $value > $pin_path/direction
            ;;

            *)
                echo "Usage: $0 mode [pin] [in|out]"
                echo "       $0 read [pin]"
                echo "       $0 write [pin] [0|1]"
                echo "If GPIO_PINS is an environment variable containing"
                echo "a space-delimited list of integers, then up to 17"
                echo "logical pins (0-16) will map to the physical pins"
                echo "specified in the list."
            ;;
        esac
    }

    if [ "$BASH_SOURCE" == "$0" ]; then
        gpio $@
    fi
```

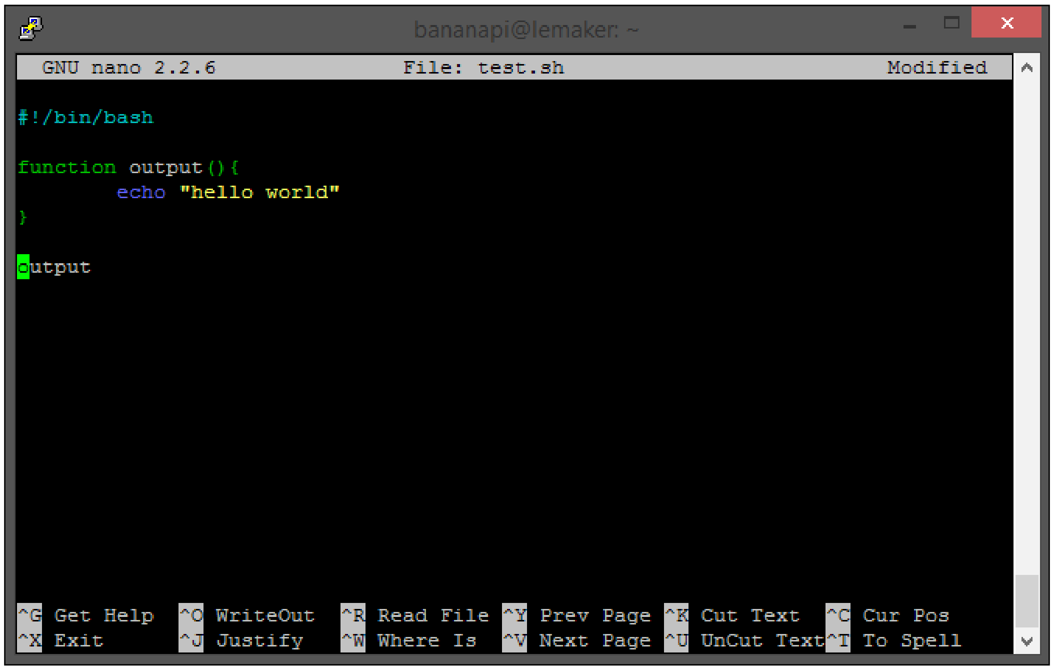
If you add that code to a file called `gpio.sh`, you can save it and then run it via the command line by passing the file into the bash command, which will tell bash to start reading the script. Scripts are read in the top-down manner, but functions such as the `gpio` function are reusable code, which won't be executed until it is called. This will cause the script to spit out the generated output:

```
bash gpio.sh
```

Functions need to be defined before they are called. Otherwise, you will get an error. If you try to execute the following script, it will blow up:

```
#!/bin/bash
output

function output(){
echo "hello world"
}
```

This is because you are trying to call the output function, but the function hasn't been read by bash yet. So, it has no idea what you are doing. If you flip them around, you will be able to run it. The following screenshot shows the bash script being written in the nano editor:



```
  GNU nano 2.2.6              File: test.sh                          Modified

#!/bin/bash

function output(){
        echo "hello world"
}

output
```

Python programming

Python is one of the most popular languages on Linux. Basically, all Linux developers channel the serpent god. Python is a great general-purpose language, which is installed by default on Linux and is available for you on the Pi right now. You can verify the version of Python that you are using by running a simple command, as follows:

```
python --version
```

You will see some output:

```
Python 2.7.3
```

The output shows that I am running Python 2.7.3, which is not the latest version of Python, but it is good enough for now. The best thing about everything is that you already got a taste of Python in the previous chapter, when we connected to the GPIO pins. The Python files on the system will have the `py` file extension. You can start playing with Python by running the `python` command in the terminal. This will start Python and drop you into a Python prompt, which will look like the following:

```
Python 2.7.3 (default, Mar 18 2014, 05:13:23)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

From here, you will be able to start writing the Python code. For an example of a Python script, you can refer to the previous chapter.

The Ruby language

Ruby is a programming language that is similar to Python. It is also installed on Linux by default. You can check out your Ruby version by running the following command:

```
ruby -v
```

You will see some output that identifies the version of Ruby that you are currently using. If you are using the **Ruby Version Manager** (**RVM**) to manage your Ruby versions, you can use the following code to check which version you are currently using:

```
ruby 1.9.3p194 (2012-04-20 revision 35410) [arm-linux-eabihf]
```

Ruby uses packages called Gems, which can be installed to give your programs additional functionality. There is a Gem called Pi Piper, which can be added for event-driven programming to the Banana Pi GPIO pins. To install it, you will need to install the `ruby-dev` package:

```
sudo apt-get install ruby-dev
```

If you don't install this package, it will fail when you try to add the Ruby Gem. To add the Gem, run the following code:

```
sudo gem install pi_piper
```

At this point, you will be able to add the `pi_piper` Gem to your script and interact with the GPIO pins. The following script is an example that demonstrates how the Ruby code interacts with the GPIO pins, waits for the detection of the pin's state, and then provides the results via the output:

```
#/usr/bin/env ruby
require 'pi_piper'
include PiPiper

watch :pin => 23 do
  puts "Pin changed from #{last_value} to #{value}"
end

#Or

after :pin => 23, :goes => :high do
  puts "Button pressed"
end

PiPiper.wait
```

You can run Ruby scripts from the command line by using the `ruby` command. If you saved the preceding code as `gpio.rb`, you can run it by issuing the following command:

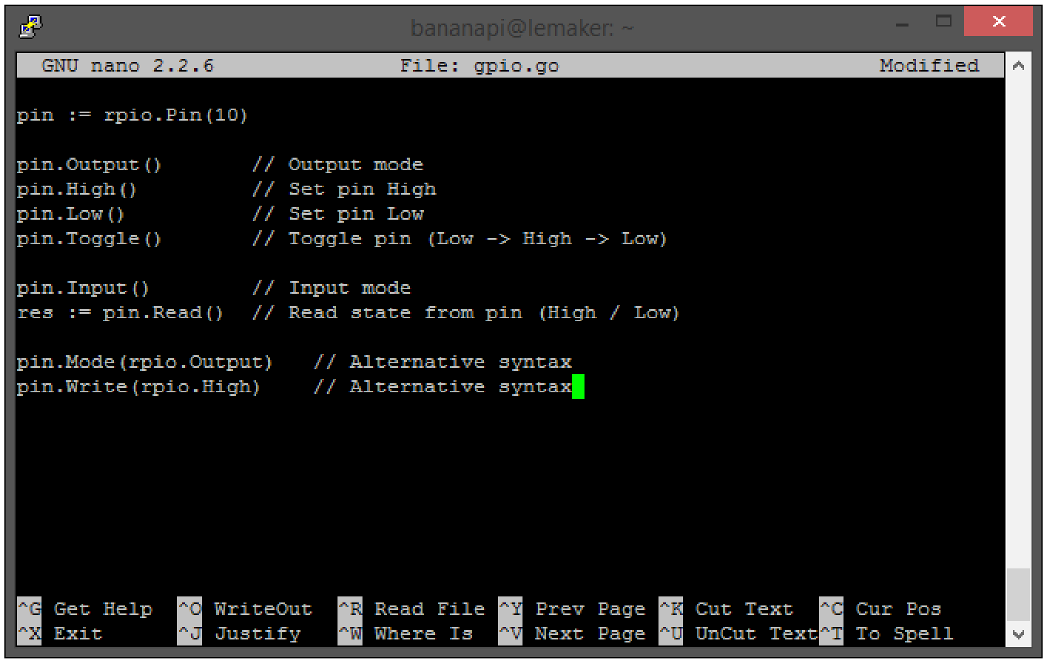
```
ruby gpio.rb
```

The preceding code will tell Ruby that you want it to read and run this file. The code will also cause it to generate any output you have decided it should spit out and in this case, wait for you to press a button on pin 23 on the Pi.

Ruby and Python are almost interchangeable. The language that you choose from the two boils down to your preference. Both have their respective pros and cons, but ultimately, it is nice to be able to use both.

The Go language

Go is Google's answer to C. It is a general compiled language that is suitable for a lot of different projects, but it is also quite fast and lightweight. The following screenshot shows Go code being written in Nano:



```
  GNU nano 2.2.6              File: gpio.go                          Modified

pin := rpio.Pin(10)

pin.Output()        // Output mode
pin.High()          // Set pin High
pin.Low()           // Set pin Low
pin.Toggle()        // Toggle pin (Low -> High -> Low)

pin.Input()         // Input mode
res := pin.Read()   // Read state from pin (High / Low)

pin.Mode(rpio.Output)   // Alternative syntax
pin.Write(rpio.High)    // Alternative syntax
```

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

Go does not come installed by default on the Pi, and it requires some setup before you can start working with it. Before we begin, we will need to install the prerequisites, as follows:

```
sudo apt-get install -y mercurial gcc libc6-dev
```

Mercurial is similar to Git (although not as cool, am I right?) We will use it to clone the remote repository onto the Pi so that we can compile the language, as follows:

```
hg clone -u default https://code.google.com/p/go $HOME/go
```

This may take a while, but will clone the Go repo into the `/go` directory inside your `home` directory. If you remember from the previous chapter, you will be able to find this in `/home/bananapi/go`. If all goes well, you will see some output that looks like the following:

```
requesting all changes
adding changesets
adding manifests
adding file changes
added 21925 changesets with 76806 changes to 11602 files (+9 heads)
updating to branch default
4441 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

Once the repo is cloned, we will need to compile the source. First, we will need to change the directories to the source directory, as follows:

```
cd ~/go/src
```

Now we will need to compile the source, as follows:

```
./make.bash
```

Compiling will take a while. You can check your e-mail or make a cup of coffee while you wait. Once the Go language has been compiled, we can add it to our path so that we can easily call it, as follows:

```
export PATH=$PATH:$HOME/go/bin
```

You should now be able to check the Go version by running the following version command:

```
go version
```

There is a Go library for the Pi that allows you to interact with the GPIO pins just like we did with Python and Ruby. This package is called Go-RPIO, and it can be added to your project by including it from the beginning, as follows:

```
import "github.com/stianeikeland/go-rpio"
```

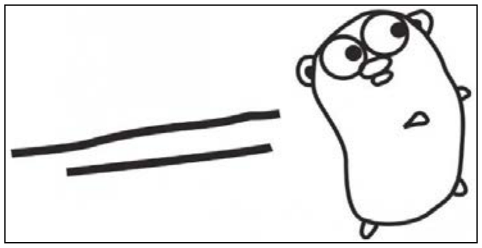
This will allow you to write the Go code and interact with the Pi. This is a script that will toggle an LED on pin 19. We will import the proper libraries first in the following way:

```
package main

import (
  "fmt"
  "github.com/stianeikeland/go-rpio"
  "os"
  "time"
)
```

Now we can proceed to define our pin and write our main function, which will toggle the LED, as follows:

```
var (
  pin = rpio.Pin(10)
)

func main() {
  if err := rpio.Open(); err != nil {
    fmt.Println(err)
      os.Exit(1)
  }
  defer rpio.Close()
  pin.Output()
  for x := 0; x < 20; x++ {
    pin.Toggle()
    time.Sleep(time.Second / 5)
  }
}
```

You do not need any other libraries to use the GO-RPIO library. However, we used `fmt`, `os`, and `time` because we are working with time and output. Go is a fast language. Therefore, once it is compiled, it can be executed quickly. Its logo is a Gopher that moves quickly. It can be illustrated in the following way:



Summary

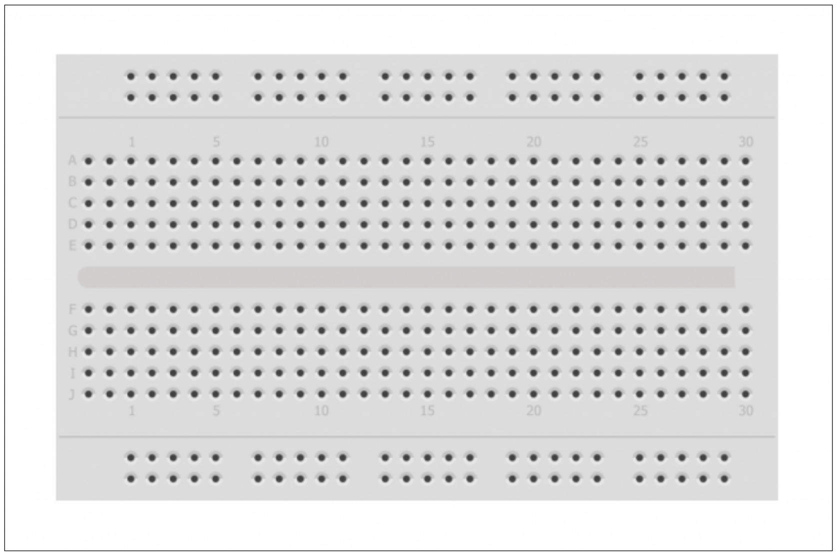
In this chapter, we explored several different programming languages, command-line tools, graphical editors, and even some web IDEs. These tools are valuable for all kinds of projects that you may be working on. Whether you are building an Internet of Things device or a small code server to learn how to code using one of the Web IDE solutions, you can now accomplish all of this.

In the next chapter, we are going to start playing with the various hardware aspects of the Pi, which includes working with LEDs, controlling motors, and connecting your Pi to a breadboard. We will also write more code to do all of these things in the different languages that we have learned in this chapter.

5
Hardware for Your Pi

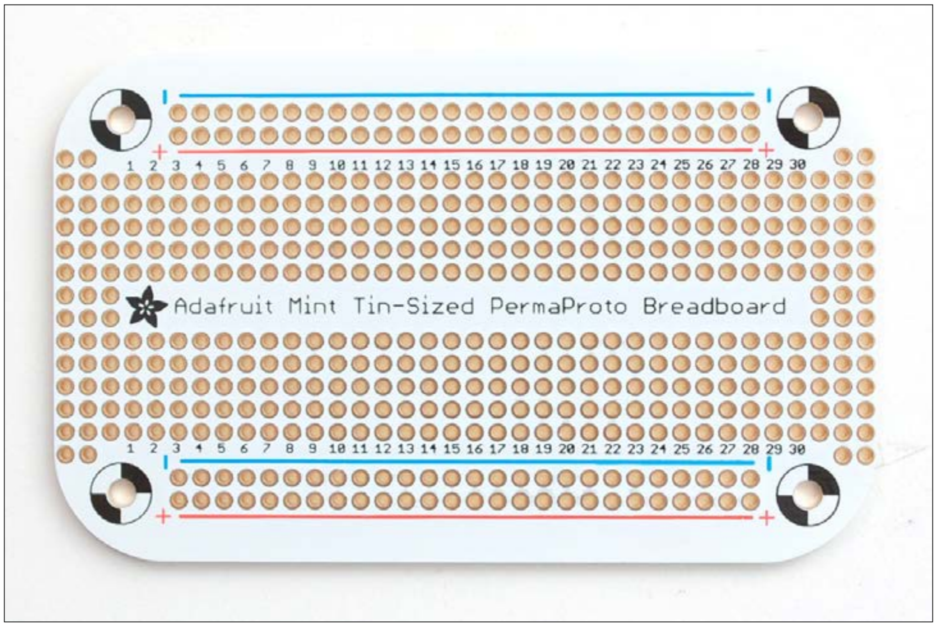
Now that we have explored working with some of the programming languages that are available to you on the Banana Pi, we can start getting into the hardware side of things. We can write the code now that we will use to interact with LEDs and motors. In this chapter, we will talk about breadboarding your Pi and some of the components that you can use in your projects.

Hardware prototyping is made easy when you have a tool like the Banana Pi. In the previous chapters, we wrote code that interacted with the GPIO pins. Now we are going to use these pins to do some fun stuff. There are some things that we need to cover first, such as what hardware is required for this and how to properly use the hardware. The following screenshot shows a breadboard:



Breadboarding your Banana Pi

The breadboarding of electronics essentially involves the prototyping of what you will eventually be able to build into a complete product. The breadboard acts as your circuit board. It will have the components attached directly to it. There are multiple kinds of breadboards, but all of them perform the same task. Some breadboards are soldered (as shown in the following picture)—which means that the components added must be soldered to the board—while some are not:

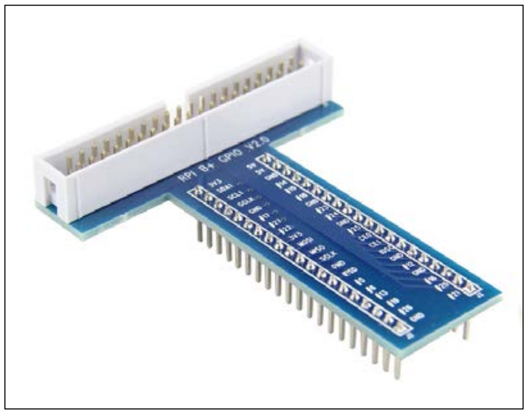


For the purposes of learning, in this book I will use a solderless breadboard. This will allow us to attach components directly to the board without soldering.

Required hardware

So you have your Banana Pi and its shiny GPIO pins, and you have a breadboard. However, now you are standing there thinking, "How do I put these things together?"There are a few ways to breadboard the Banana Pi. There are a couple of accessories available that are intended to breadboard the Banana Pi, but you don't need them.

The following image shows the GPIO breakout for the Banana Pi M1, which is also compatible with the Model B Raspberry Pi:



In addition to the breakout, you will need a ribbon cable that will attach to the GPIO pins on one end and the breakout on the other. This will allow you to connect the Banana Pi directly to the breadboard and start working with the components. The M1 board has 26 GPIO pins, while the M2 has 40.

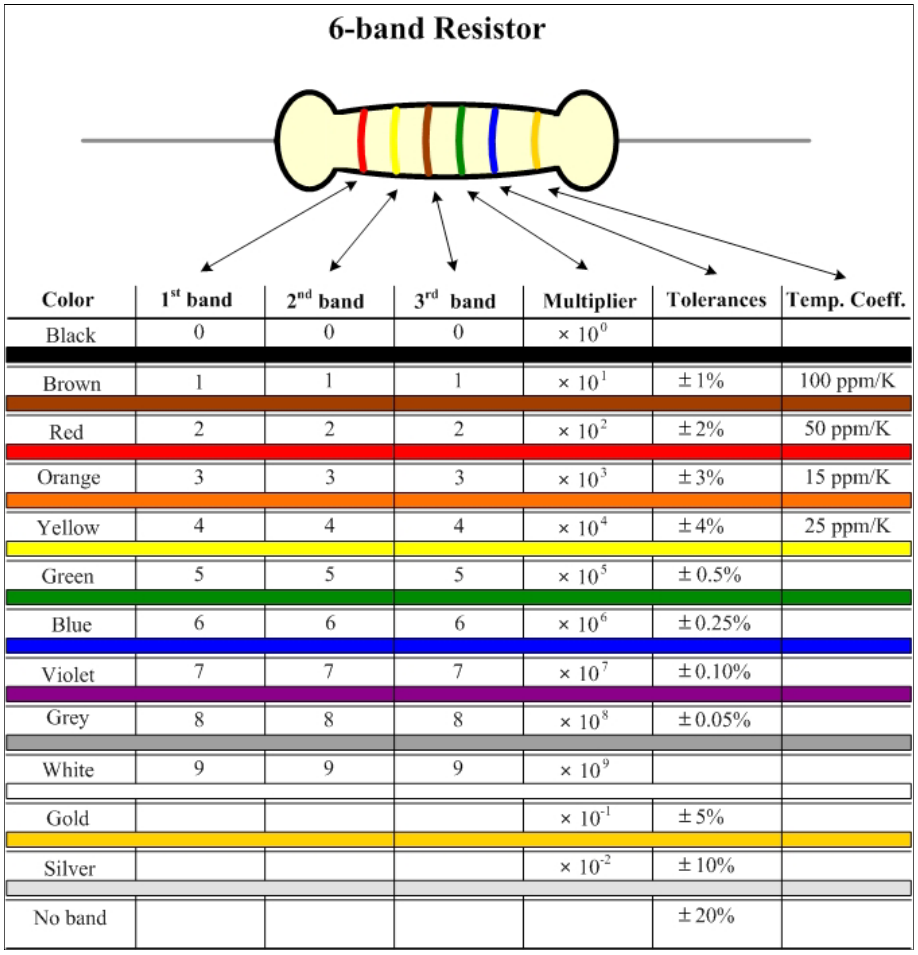
Common components

Regardless of your project, you will find yourself working with some common electrical components. There are many different components that you can use in your circuits. We won't cover them all, but we will cover some common ones that you might use when you are prototyping. These components exist in many projects, and you will need to become familiar with them.

Resistors

Resistors are components that resist the flow of the current. This is a very basic component that is used in most circuits and is very important. Resistors come in a wide range of values; the value determines how much flow of electricity they resist. The resistance is measured in Ohms, which is represented by the Ω symbol. The amount of power the resistor can handle before burning to a crisp is measured in Watts.

The resistor is shown as a zigzag or a rectangle on a schematic, and it is present in almost every circuit. Resistors can be made out of many different kinds of materials, but they are most commonly made of carbon. The following image shows how you can tell how much resistance each resistor offers:

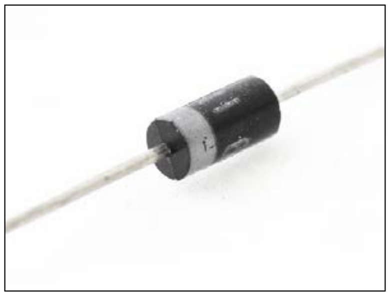


6-band Resistor

Color	1st band	2nd band	3rd band	Multiplier	Tolerances	Temp. Coeff.
Black	0	0	0	$\times 10^0$		
Brown	1	1	1	$\times 10^1$	± 1%	100 ppm/K
Red	2	2	2	$\times 10^2$	± 2%	50 ppm/K
Orange	3	3	3	$\times 10^3$	± 3%	15 ppm/K
Yellow	4	4	4	$\times 10^4$	± 4%	25 ppm/K
Green	5	5	5	$\times 10^5$	± 0.5%	
Blue	6	6	6	$\times 10^6$	± 0.25%	
Violet	7	7	7	$\times 10^7$	± 0.10%	
Grey	8	8	8	$\times 10^8$	± 0.05%	
White	9	9	9	$\times 10^9$		
Gold				$\times 10^{-1}$	± 5%	
Silver				$\times 10^{-2}$	± 10%	
No band					± 20%	

Diodes

Diodes are components that allow the current to flow in only one direction. The diode has two terminals, called the anode and cathode. The current flows through the diode only when a positive voltage is applied to the anode side and a negative voltage is applied to the cathode. If you try to make the current flow the other way, it will not. The following image shows a diode:

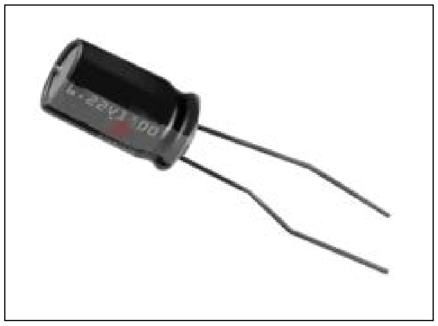


Light-emitting diodes

Light-emitting diodes (**LEDs**), just like normal diodes, are components that allow the current to flow in only one direction. When the current passes from the anode to the cathode, the diode emits light. The following screenshot shows the different types of LEDs:

Capacitors

A capacitor is a very common component that is used to temporarily store electricity. The capacitor is used to smoothen the output of power supplies and block direct current. It can also be used to tune radios to frequencies. The following image shows a capacitor:



Transistors

Transistors are three-terminal devices that can control the flow of electricity across two of its terminals when a voltage is applied to one of the terminals called the base. The transistor is one of the most important components in electronics.

Integrated circuits

An integrated circuit contains an entire electric circuit. It has its own tiny diodes, transistors, resistors, and so on. These components are one of the most important components in modern electronics. The components are etched onto tiny pieces of silicon and can be used for many different purposes. Sometimes, these chips are referred to as bug chips. The following image shows an integrated circuit:

An integrated circuit can have several billion transistors in them, all the size of a fingernail. This is an impressive piece of technology that has made a lot of innovations possible. Integrated circuits allow you to have a lot of functionality added to your circuit for a fraction of the cost.

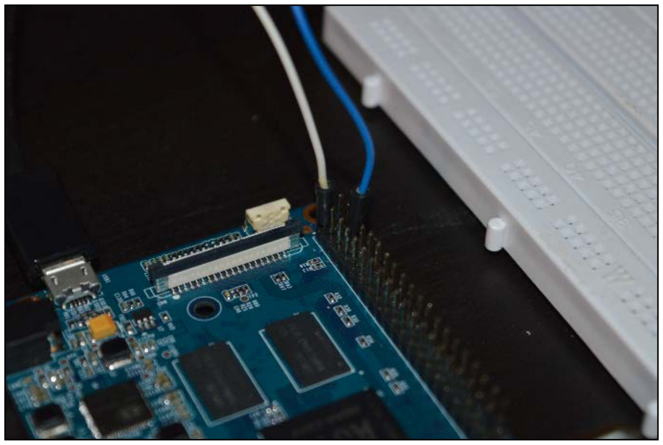
Working with hardware

Now that we have covered the method of breadboarding your Pi and some of the electrical components that you might use in your circuits, we can talk about putting these into practice with our Banana Pi. We will start with some simple circuits and move on to the process of controlling motors through the GPIO pins.

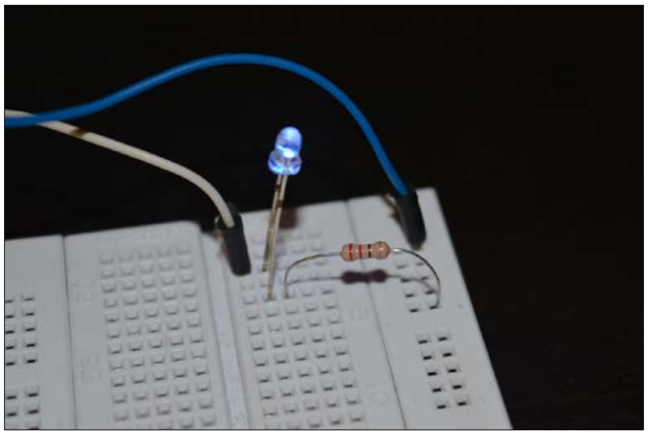
Working with LEDs

The simplest circuit that we can make (aside from just a little wire!) is lighting up an LED. You can consider this as the "Hello World" of electric circuits. For this, we are going to need an LED (any color), some jumper wires (two pieces), and a resistor. The resistor needs to be between 270 Ω and 330 Ω. If you go higher than that, the LED will be dimmer than usual. With too much current, the LED will be very bright before it burns right out.

This is a very simple circuit. We will connect a wire to the 3.3V pin and the ground pins. It doesn't matter which Banana Pi you are using, though you may need to refer to the chart from the first chapter to reference the pins. I am using the M2 for this example. The following image shows the pins that I am using:



I have created a simple circuit by connecting a white jumper to the 3.3V pin (top left) on the M2. This pin goes to the breadboard and is plugged into the same rail as the anode of an LED. This LED leads to a resistor, which prevents excessive current from passing through the circuit. The blue jumper is connected to the other end of the resistor on the negative rail, which is plugged into the ground pin on the Banana Pi. In the next image, you can see the circuit that I created:



Note that we were able to create this circuit without the use of code. We can enhance this circuit by writing a bit of code to make the LED blink.

Controlling LEDs with code

Now that we have created a simple circuit using our Banana Pi, we can interact with it by using a bit of code. We will revisit the Python coding language that we talked about in a previous chapter. Let's start by moving the jumper on the 3.3V pin and move it to pin 12, which represents GPIO 21. You can refer back to the chart if you need to, but it is the 7th pin down the left side of the board.

We need to create a simple script. Let's start by importing what we will need to use the time package and the GPIO pins:

```
import RPi.GPIO as GPIO
import time
```

We will need a function to blink a pin. We will fluctuate between turning the LED on and off every second by using the following code:

```
def blink(pin):
  GPIO.output(pin,GPIO.HIGH)
  time.sleep(1)
  GPIO.output(pin,GPIO.LOW)
  time.sleep(1)
  return
```

This simple function will be the section of our code that actually handles the current that is being supplied to the LED. Now, we need to set up the pins and the mode. We will accomplish this with the help of the following code:

```
GPIO.setmode(GPIO.BOARD)
GPIO.setup(21, GPIO.OUT)
```

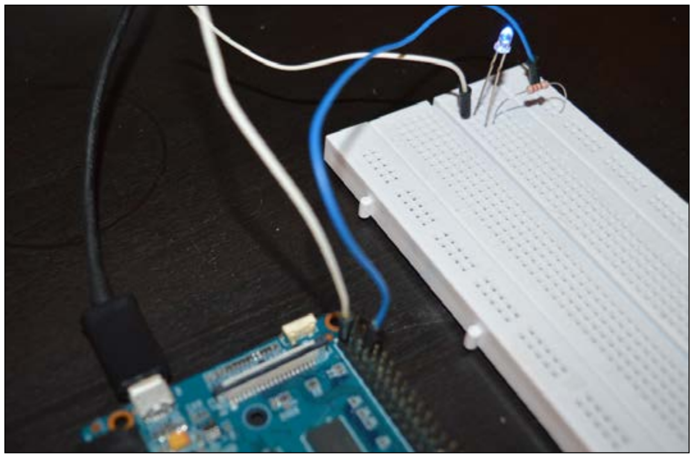
Now that our pins are initialized, we want to loop a few times over the blink function to make the LED actually blink. If we just call the function once, we will only see the LED turn on and then turn off. So, let's set up a `for` loop that will iterate 50 times, calling our function each time:

```
for I in range(0,50)
  blink(21)
```

Now, we will call the `cleanup()` function with the help of the following code:

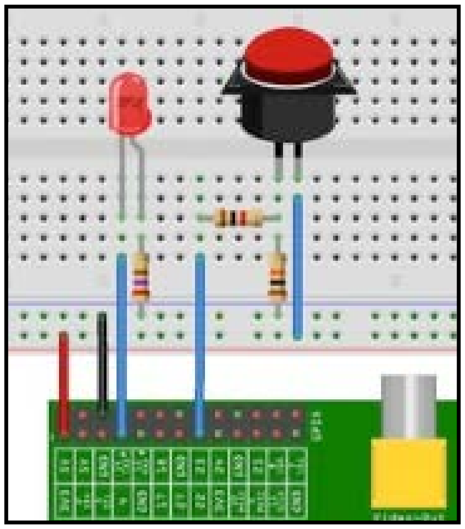
```
GPIO.cleanup()
```

The preceding function will clean up all the ports that we used. This will only affect the pins that we used in our program (21). This is because programs don't necessarily end just because they have hit the bottom of the script. Our pin will still be initialized after the loop, which means that it will still be in the memory. By calling this function, we can reinitialize the pin later on when we need to use it again. Here is a picture of my circuit:



Adding user input

Now, we have a simple circuit that can be controlled by code. Let's add some user input to our project. We will connect a button to the breadboard and use the input to light up our LED. In the end, our circuit will look like this:

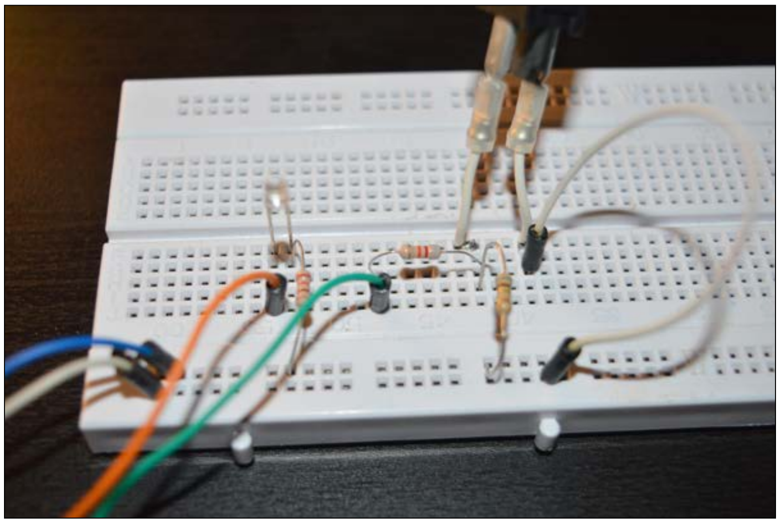


We are going to need the following additional hardware:

- Two additional jumper wires
- A button
- A 1K Ohm resistor
- A 10K Ohm resistor

We will start by connecting the LED to the 270 Ohm resistor and the 3.3V input. We are going to connect the button to the pin numbered 22 and a 10K Ohm resistor. When the button is pressed, the circuit diverts the power from the 3.3V pin to the pin numbered 22, which bypasses the 10K Ohm resistor. The 1K resistor is there to protect the Banana Pi.

When all of this is on the breadboard, it will look like this:



Now, we will write some more code in Python to make the LED light up when the button is pressed.

So, in this script, we are going to make the LED on pin 21 blink when the button on pin 22 is pressed. To start out, we will import the packages that we need to start our project just like before, in the following way:

```
import RPi.GPIO as GPIO
import time
```

Now, we are going to set up the GPIO pins that we will be working with. Again, this will be similar to our last script, except for the fact that, in this case, we will be using two pins:

```
#Set up GPIO
GPIO.setwarnings(false)
GPIO.setmode(GPIO.BCM)

#Set up pins
GPIO.setup(21, GPIO.OUT)
GPIO.setup(22, GPIO.IN)
```

We will set up an infinite loop, which will set the value of the LED to the button being pressed, with the help of the following code:

```
While True:
  GPIO.output(21, GPIO.input(22))
  time.sleep(0.05)
```

The preceding code will cause the LED to light up when the button is pressed. You can save this script and run it with Python on the Banana Pi. Now try pressing the button.

We have pretty much mastered the art of working with LEDs. We will now move on to something more fun.

Working with servos

One of the most popular things that you can work with now is servo motors. Servos are small electric motors that are popular in robotics and several other related projects. We will write a small script that will allow us to control a servo motor.

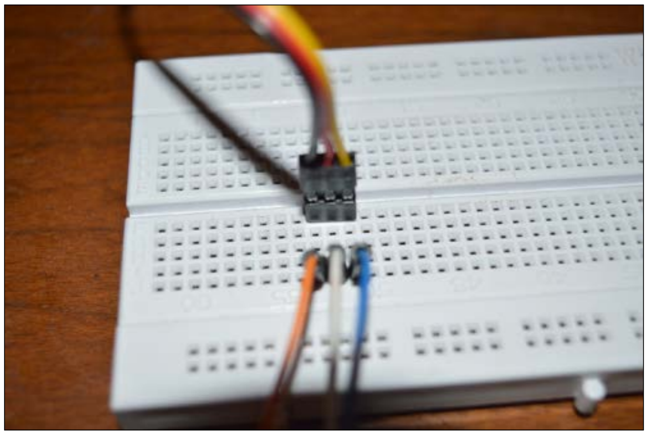
The following image shows a servo motor:

We will use the Pi to control the position of the servo. For this project, we are going to need the following components other than your Pi:

- A servo motor
- Three jumper wires
- A breadboard

We will connect the servo to three pins on the Banana Pi. Unlike a stepper motor, we can safely connect the stepper directly to the Pi without having to use a bunch of resistors or integrated circuits. We will also write a simple script that will be a little more complex than our previous scripts. With this script, we will be able to directly control the stepper motor.

First, we will connect the red wire (servo +5V) to the breadboard. On the same rail, we will connect a wire to pin 4 on the Banana Pi. Next, we will connect pin 6 on the Pi to the breadboard, which will then be connected to the black wire on the servo (-5V). Pin 11 is our PWM signal. This will be connected to the yellow wire from the servo. When we are done, the circuit will look like this:



Now that we are all wired up, we can write our Python script.

Controlling our servo

We have our servo connected to our Pi. We want to control the position of the servo by using a script. We will start by writing a new script:

```
nano servo_controller.py
```

The first thing we want to do is import our packages and set up the GPIO pins like our previous scripts, as follows:

```
import RPi.GPIO as GPIO
import time

# Set up GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(11, GPIO.OUT)
```

Now let's output some instructions so that we know what keys to press in order to move the servo. This is optional, but we will add it in case we forget which keys to press, as follows:

```
print "l : Left"
print "r : Right"
print "m : Middle"
print "q : Quit"
```

Now, we need to add some code that will listen for the correct keys and act appropriately. If the *L* key is pressed, we will move the servo to the left. If the *R* key is pressed, we will move the servo to the right. We can also move the servo to the center position and quit our program.

First things first. We need to set up an infinite loop that will run and listen for the keys that we will be pressing. We also need to define the servo (we will define that on pin 11), as follows:

```
While True:

  Servo = GPIO.PWM(11, 50)
  Servo.start(2.5)

  input = raw_input("Selection: ")

  if(input == "r"):

  elseif(input == "l"):

  elseif(input == "m"):

  elseif(input == "q"):

  else:
    print "Input not recognized"
```

Now, we need to handle the different keys that are pressed. Right now, we have a script that listens for input and then filters the script down to a specific piece based on which key was pressed.

First, we will handle the *R* key when it is pressed. We want to see how many steps we have to take towards the right based on the input from the user. We will then loop the function as many times as we are told and move the servo to the right. We will receive input from the user, as follows:

```
if(input == "r"):

  steps = raw_input("Steps (1 - 10): ")
  steplength = 12.5 / int(steps)
```

Now, we will iterate as many times as the user has input and move accordingly, as follows:

```
for Counter in range(int(steps)):
  Servo.ChangeDutyCycle(stepslength * (Counter + 1))
  Print stepslength * (Counter + 1)
  time.sleep(0.5)

time.sleep(1)
Servo.stop()
```

We will handle the process of moving towards the left next. This is similar to what we just did:

```
elseif(input == "l"):
  Servo.start(12.5)
  steps = raw_input("Steps (1 - 10): ")
  stepslength = 12.5 / int(steps)

  for Counter in range(int(steps))
    Servo.ChangeDutyCycle(12.6 - (stepslength * (Counter + 1)))
    time.sleep(0.5)

  time.sleep(1)
  Servo.stop()
```

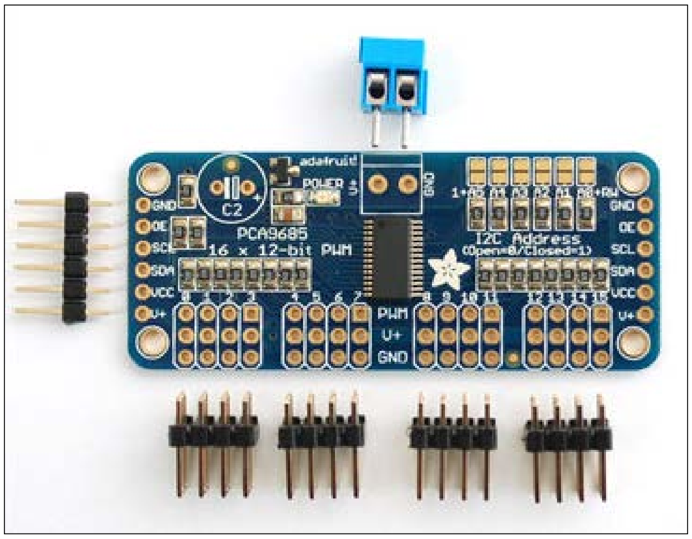
Next, we need to handle *M*. This will move the servo back to the middle position. This is almost like a reset switch for our script, which can be implemented in the following way:

```
elseif(input == "m"):

  Servo.start(7.5)

  time.sleep(1)

  Servo.stop()
```

Now, we will handle exiting the script and cleaning up the GPIO pins when the user selects *Q* in the script, as follows:

```
elseif(input == "q"):

  os._exit(1)

  Servo.stop()

  GPIO.cleanup()
```

Now, when you run the script, you will be presented with the selections. Based on your input, the Banana Pi will control the servo and move it in the direction of your choosing. This is our first interactive script. You can apply what we did here to all kinds of sensors and components, which will be covered in the next chapter. You can take things a step further and control multiple servos by using a servo controller, such as the one shown in the following image:



Our complete script looks like this:

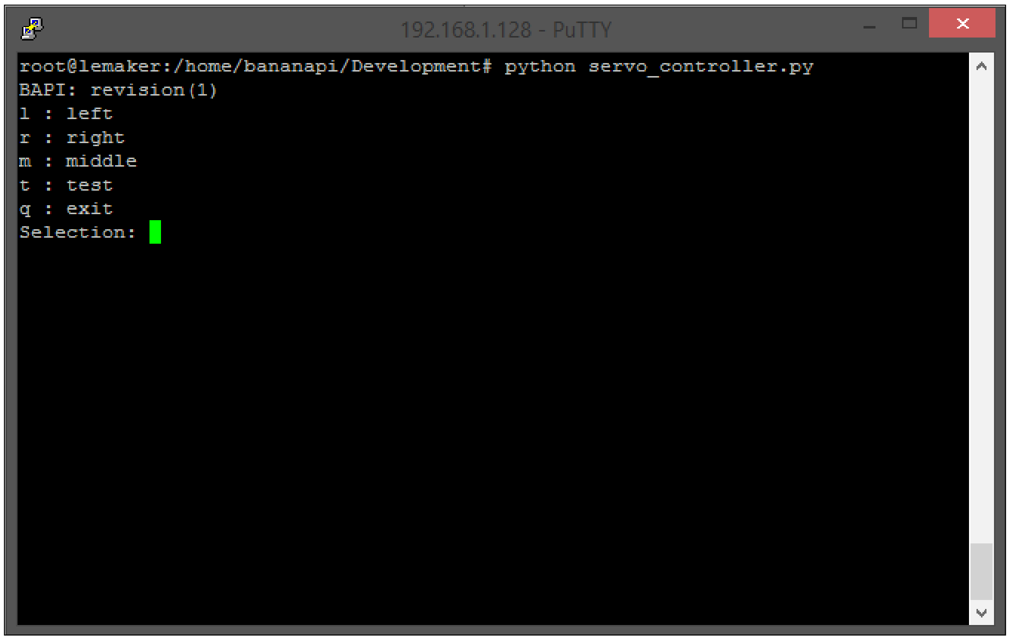
```
import RPi.GPIO as GPIO
import time
import os

# Set up the GPIO
```

```
GPIO.setwarnings(False)

GPIO.setmode(GPIO.BOARD)

GPIO.setup(11, GPIO.OUT)

print "l : left"

print "r : right"

print "m : middle"

print "t : test"

print "q : exit"

while True:

  Servo = GPIO.PWM(11, 50)

Servo.start(2.5)

  input = raw_input("Selection: ")

  if(input == "r"):

    steps = raw_input("steps (1 - 10): ")


    stepslength = 12.5 / int(steps)

    for Counter in range(int(steps)):

      Servo.ChangeDutyCycle(stepslength * (Counter + 1))

      print stepslength * (Counter + 1)

      time.sleep(0.5)

    time.sleep(1)

    servo.stop()
elif(input == "m"):

    Servo.start(7.5)

    time.sleep(1)

    Servo.stop()

  elif(input == "l"):

    Servo.start(12.5)

    steps = raw_input("steps (1 - 10): ")

    stepslength = 12.5 / int(steps)

    for Counter in range(int(steps)):

      Servo.ChangeDutyCycle(12.5 - (stepslength * (Counter + 1)))
      time.sleep(0.5)

    time.sleep(1)

    Servo.stop()

  elif(input == "q"):

    os._exit(1)

    Servo.stop()

    GPIO.cleanup()

  else:

    print "Input not recognized"
```

You can save this as `servo_controller.py` and run it from the command line by using the following command:

```
python servo_controller.py
```

The following screenshot shows the output of the program that we just wrote:



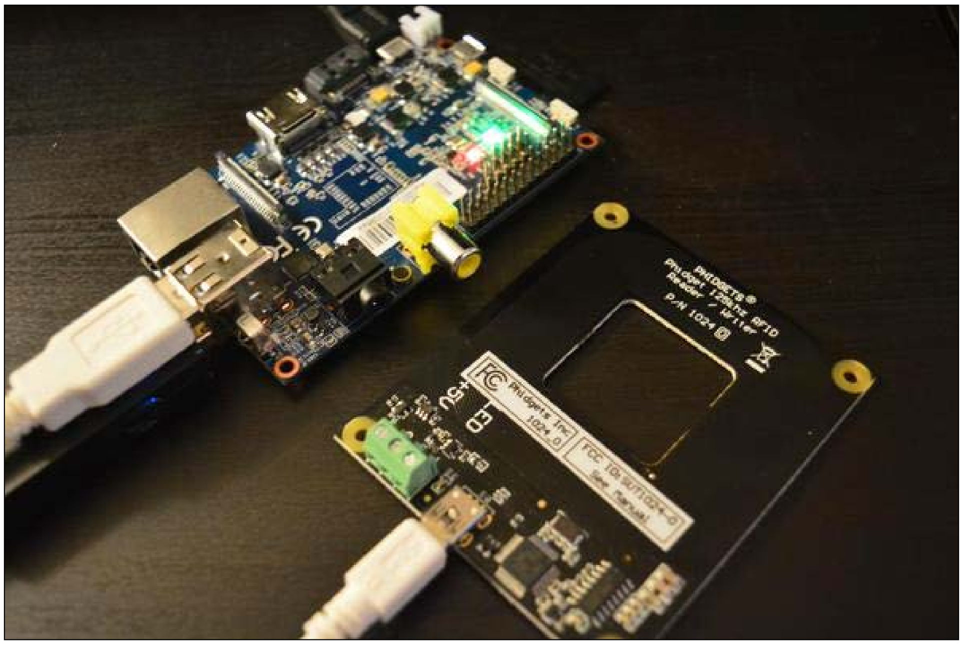
```
root@lemaker:/home/bananapi/Development# python servo_controller.py
BAPI: revision(1)
l : left
r : right
m : middle
t : test
q : exit
Selection:
```

Summary

In this chapter, we worked with LEDs and servos. We controlled them with the code written directly in the Banana Pi and interacted with the hardware both through buttons and input on the script. Now, you should have started to see the potential of what can be accomplished.

In the next chapter, we will cover the process of interacting with sensors. We will explore infrared receivers that can be used in a remote control to control your Banana Pi. We will also explore the operation of prototyping hardware such as Phidgets, which can extend the Banana Pi, and take advantage of hardware such as RFID and light sensors to control the motors.

The following image shows the Phidgets RFID reader connected to the Banana Pi:

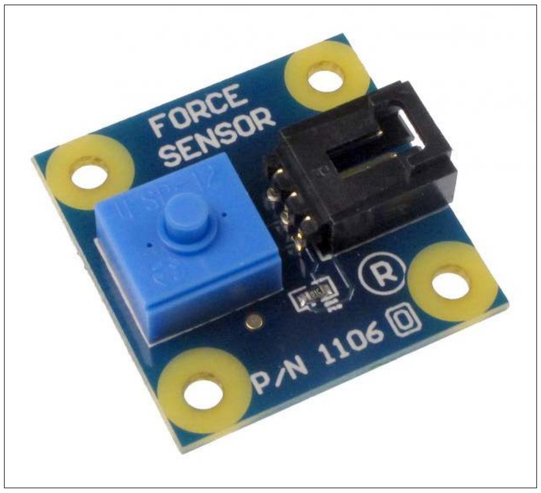


We will also explore the interaction with GPIO-connected hardware from the input over the USB.

6
Interacting with Sensors

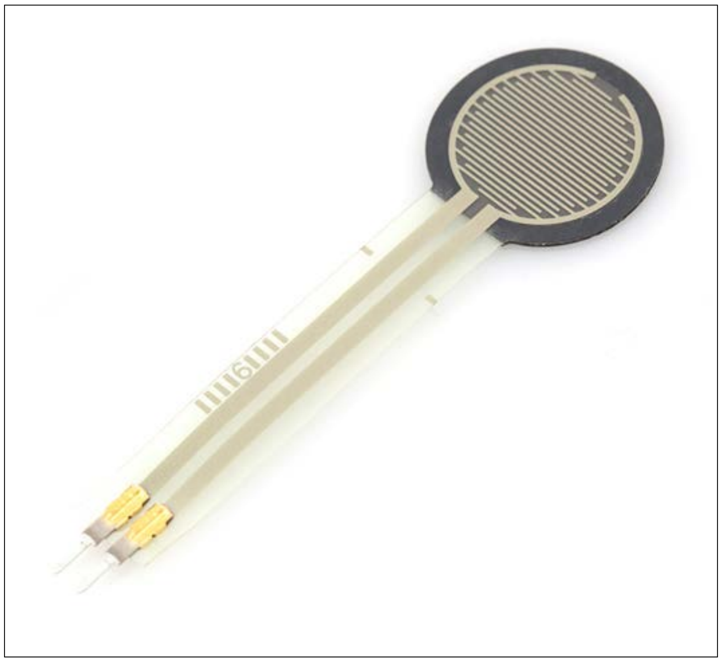
Playing with the LEDs and hardware is awesome, but we can now kick it up a notch. Basically, the projects that you want to do—aside from the ones related to software—are likely going to have some sort of a sensor incorporated into it. This is common regardless of whether you are building a prototype or doing some home automation.

Sensors are components that allow us to interact with the physical world. They do different things based on what you want to detect. A light sensor can tell you how much light it is being exposed to, and force sensors can tell you how much force is being applied to them. There are more types of sensors than we could ever hope to cover in this book. Using sensors allows us to write code that can trigger events that depend on something that physically happens. For example, if there is enough light coming through your window, a light sensor can trigger your blinds to open. You can use an RFID card to not only unlock something, but also trigger an e-mail to you. The following image shows a force sensor by Phidgets:



Sensors and you

There are a lot of different sensors from a lot of different brands. Brands such as Phidgets are like small prototyping boards in themselves, and there are components that you can add directly to the Banana Pi. The force sensor that was shown previously functionally does the same thing as the following sensor:

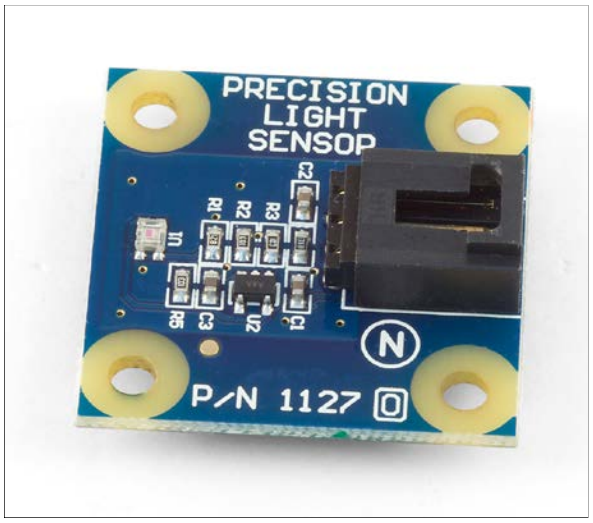


Although sensors such as Phidgets require additional software, the setup is generally not that difficult. Since we have been covering programming languages such as Ruby, it won't be difficult to set up the packages.

We can start making some really interesting stuff, especially when we introduce other prototyping hardware such as Phidgets. Although Phidgets are interfaced through USB, we can write a single program that controls the hardware both over GPIO and USB. The scripts we write don't care where the hardware is connected as long as we define it first.

Interacting with light

One of my favorite sensors to use in projects is a precision light sensor. What this will do is send a signal when it detects light. We will use the precision light sensor by Phidgets. However, you don't need to use this particular brand. The following image shows a light sensor:



To get started, we will need to install the Phidgets driver and the RubyGems required to interact with the sensors. We will then write a generic Ruby script that can work with most of the sensors and that we can modify to do other stuff later. This script will look for sensors that are connected to a main control board and then move a servo. First, we will install the `libusb` development libraries by using the following command:

```
sudo apt-get install libusb-1.0-0dev
```

Now we will need to download and build the Phidgets libraries. This will allow us to actually work with the sensors. You can grab the file from `http://www.phidgets.com/docs/OS_-_Linux`.

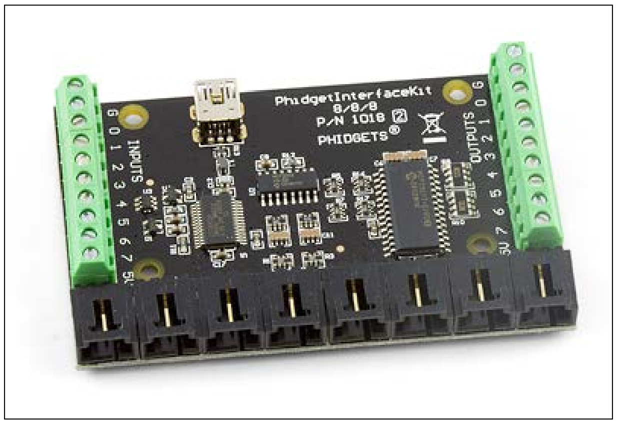
You will need to unpack the library and put it wherever you want. I unpacked it in the home directory. Also, you will need to change the directories to refer to the Phidgets library and compile it, as follows:

```
./configure
make
sudo make install
```

These libraries are written in C. The other libraries that we will need (for example, Ruby) will depend on them to get the job done. Now that we have the library installed, we will need to install a couple of Ruby gems (we will use Ruby for these programs). We can install the gems needed with a couple of commands, which are as follows:

```
sudo gem install ffi
sudo gem install Phidgets-ffi
```

Now we can start interacting with the sensors by using the Phidgets controller. The following image shows the controller board that we will use; this interfaces with the Banana Pi over USB and has pins to connect directly to sensors, such as precision light sensors:



We will start our script by using the `require` function of Ruby with the gems that we installed. These gems will be used to actually interface with the sensors, as follows:

```
#!/usr/bin/ruby
require 'rubygems'
require 'phidgetsffi'
```

Now we can run setup and define our sensors. We will use the advanced light sensor and an advanced servo motor. The servo motor can be replaced by anything you like, even a servo that is controlled directly by the GPIO pins, such as the one that we saw in the last chapter. Here's the script:

```
puts "Library Version: #{Phidgets::FFI.libray_version}"

ifkit = Phidgets.InterfaceKit.new
adv = Phidgets.AdvancedServo.new
```

Now we can set up the servos, as follows:

```
adv.on_attach  do |device, obj|
  puts "Servos Initialized"
  device.advanced_servos[0].engaged = true
  sleep 1
end

adv.on_detach  do |device, obj|
  puts "#{device.attributes.inspect} detached"
end

adv.on_error do |device, obj, code, description|
  puts "Error #{code}: #{description}"
end
```

The methods that we just created will handle the `on attach`, `detach`, and `error` actions. These are automatically executed, depending on whether the servo is physically attached or not. We will do the same for our sensor, as follows:

```
ifkit.on_attach  do |device, obj|
  sleep 1
  if(device.sensors.size > 0)
    device.ratiometric = false
    device.sensors[0].data_rate = 64
    device.sensors[0].sensitivity = 15
    device.outputs[0].state = true
    sleep 1 #allow time for digital output 0's state to be set
    puts "Is digital output 0's state on? ... #{device.outputs[0].
on?}"
  end
end
```

We now have our `on_attach` method. This will check the device and provide an output if the state is on. We will now need to handle the action when the sensor changes. For the following example, it will be when the sensor is exposed to more light:

```
ifkit.on_sensor_change do |device, input, value, obj|

  position = value / 10
  max = adv.advanced_servos[0].position_max

  if position < max
    adv.advanced_servos[0].position = position
  end
end
```

The servo will move according to the light that the sensor receives. Since the sensor is accurate and the servo has a limited range of movement, we will take the value returned by the sensor and divide it by 10. Now we have some cleaning up to do:

```
sleep 10

# Closing the connection
puts 'DONE'
gets.chomp

# Closing the connection
puts 'DONE'
ifkit.close
adv.advanced_servos[0].engaged = false
sleep 1
adv.close
```

The preceding code will put our program to sleep, close the connections to the servo, and end the process.

Using RFID

Radio-frequency identification (**RFID**) is a popular way to transfer an ID code wirelessly by using electromagnetic fields. This technology is often put to use in projects because it is so common in the industry. The RFID tags can be used to identify employees or objects, and it is fairly inexpensive to implement. We will create a script by using the Phidgets RFID reader to move a servo, which will symbolize unlocking a door. The following image shows the RFID reader that I will be using for this example:

We will use the same Ruby gems that we imported for the last example. Note the similarities with the previous script. This shows how dynamic the code that we are using can be.

We will start by requiring the gems that we need, as we did previously. This is necessary to interact with the RFID reader and control the servo:

```
#!/usr/bin/ruby
require 'rubygems'
require 'phidgetsffi'
```

We will define the RFID reader and servo in order to use them, as follows:

```
puts "Library Version: #{Phidgets::FFI.library_version}"

puts "Waiting for Phidgets..."

# Define the phidgets
rfid = Phidgets::RFID.new
adv = Phidgets::AdvancedServo.new
```

We will now handle the servo functions, as follows:

```
adv.on_attach  do |device, obj|

  puts "Servos Initialized"
  device.advanced_servos[0].engaged = true
  sleep 1
end

adv.on_detach  do |device, obj|
  puts "#{device.attributes.inspect} detached"
end

adv.on_error do |device, obj, code, description|
  puts "Error #{code}: #{description}"
end
```

We will also need to handle the `on_attach` function for the RFID reader. This is going to enable the antenna and the LED to indicate that it is switched on. The antenna will listen for the chip and read the code that comes with it:

```
rfid.on_attach  do |device, obj|
  puts "RFID Activated"
  rfid.antenna = true
  rfid.led = true
  sleep 1
end
```

We will want to actually work with the tag when it is presented. The tag can be a card, coin, or really anything with the actual RFID chip inside it. The `on_tag` function will be as follows:

```
rfid.on_tag do |device, tag, obj|

  # Interact with the servo
  if tag == "4d004b113e"

    puts "Authorized for #{tag}"

    3.times do
      max = adv.advanced_servos[0].position_max
```

```
        adv.advanced_servos[0].position = rand(max)
        sleep 0.5
      end

    else

      puts "Tag #{tag} unauthorized"

    end

  end
```

We will also handle the `on_tag_lost` function. We won't do much in this function. We will just provide an output that notifies that the tag was removed, as follows:

```
rfid.on_tag_lost do |device, tag, obj|
  puts "Tag #{tag} removed"
end
```

We will now wait for the tag to be presented:

```
# Waiting for the RFID
puts "Present ID tag....."

gets.chomp
```

What the `on_attach` function is doing is this—when a tag is presenting to the antenna, it triggers the function. The tag is evaluated to see whether it equals `4d004b113e`. Then we will move the servo three random times.

This can be easily modified to evaluate the tag against a company database, authenticate a user, and then open a locked door. We will finish our script by closing the connection to the servo controller and RFID reader, as follows:

```
# Closing the connection
puts 'DONE'
rfid.close
sleep 2
adv.advanced_servos[0].engaged = false
sleep 1
adv.close
```

Using IR receivers

The Banana Pi has a built-in IR receiver. This is a sensor that listens for an IR signal that is sent from a device such as a remote, as shown in the following image:

This allows you to turn your Banana Pi into something like an HTPC for your TV. In order to get this to work, though, you will need some software. We will install LIRC, which will read the signal and allow us to perform different actions. If you don't install this and set the receiver as a keyboard input, you can actually write a bunch of gibberish just by using a remote.

To get started, we are going to install LIRC. We will do this via the command line, as follows:

```
sudo apt-get install lirc
```

Now, you can check out the hardware configuration file in `/etc/lirc/hardware.conf`. This file will have all the configurations for the remote, kernel support, and several other options. There are a couple of other configuration files, but they will be empty by default. We won't worry about them for now. We will now add the kernel modules to `/etc/modules`. We will add one of the following lines to the file:

```
    modprobe sun4i_ir
    modprobe sunxi_ir
```

We will need to check whether we have the correct device by running the checks for the devices that are present under `input`. We will accomplish this task by using the following command:

```
cat /proc/bus/input/devices
```

It will likely be `event1`. The thing about the IR remote is that it isn't a one-size-fits-all device. Some remotes work and some don't. I've seen people try out many different remotes only to have a couple that will work. There is a way to test, though. First, we will need to install **evtest**, as follows:

```
sudo apt-get install evtest
```

We will run evtest by passing in the receiver. If yours is `event1`, this will work. If not, just change the command, as follows:

```
evtest /dev/input/event1
```

Once you see the output coming from a remote, you have found out the one that works with your Pi. Once we have a working remote, we can configure `/etc/lirc/hardware.conf` that we mentioned previously. The configuration will look like this:

```
# Arguments which will be used when launching lircd
LIRCD_ARGS=""

#Don't start lircmd even if there seems to be a good config file
#START_LIRCMD=false

#Don't start irexec, even if a good config file seems to exist.
#START_IREXEC=false

#Try to load appropriate kernel modules
# if LOAD_MODULES=false , modules must be preloaded, i.e. during boot
(/etc/modules)
LOAD_MODULES=true
MODULES="sun4i_ir"
# newer kernel
#MODULES="sunxi_ir"

# Run "lircd --driver=help" for a list of supported drivers.
DRIVER="devinput"
```

```
# usually /dev/lirc0 is the correct setting for systems using udev
DEVICE="/dev/input/event1"

# Default configuration files for your hardware if any
LIRCD_CONF=""
LIRCMD_CONF=""
```

You may need to change the device to correctly reflect your input device, as follows:

```
DEVICE="/dev/input/event1"
```

We can configure the remote now. We will need to set up a configuration file for the remote. We will create the file, as follows:

```
nano ~/REMOTE.conf
```

We will set up the generic configuration in the following way:

```
begin remote
  name  REMOTE
  bits           16
  flags SPACE_ENC|CONST_LENGTH
  eps            30
  aeps          100
  header        9000 4500
  one           563  1687
  zero          563   562
  ptrail        563
  pre_data_bits 16
  gap          108000
  frequency    38000
  duty_cycle   33
      begin codes
      end codes
end remote
```

We will then run the `irrecord` command and supply the `.conf` file once it is configured to your remote, as follows:

```
irrecord -H devinput -d /dev/input/event1 REMOTE.conf
```

This will be your opportunity to configure the keys. Press the buttons on the remote. You may need to press the keys more than once or hold them down for more than a couple of seconds. When you are done, there will be a new configuration file saved that has the keys that you created defined in it. We will edit this file, change the name, and remove any doubled keys (the Hex code is repeated on the same line). Lastly, we will copy the new configuration file over to `lirc.conf`, as follows:

```
mv REMOTE.conf.conf /etc/lirc/lirc.conf
```

We can start LIRC by running the following quick command:

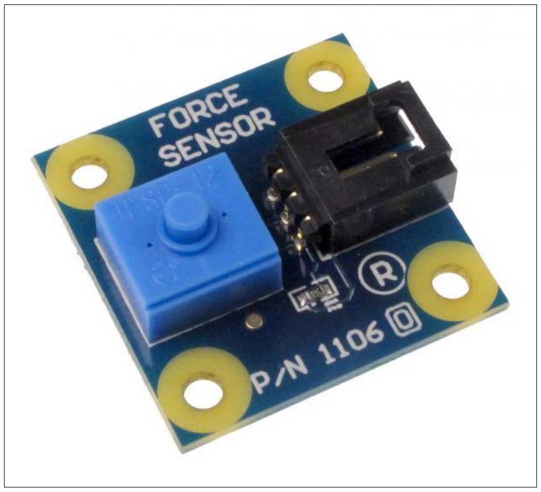
```
/etc/lirc/lirc.conf start
```

We should now have a working remote. This part is finicky because there are so many different remotes and configuration options available. Actually, I have had the most success with a remote that comes with an IR receiver, such as the one shown in the following image:

This has been my choice simply because they were made to go together, although this does use up one of your USB ports. We will write some more code to actually use an IR receiver in a project in the next chapter.

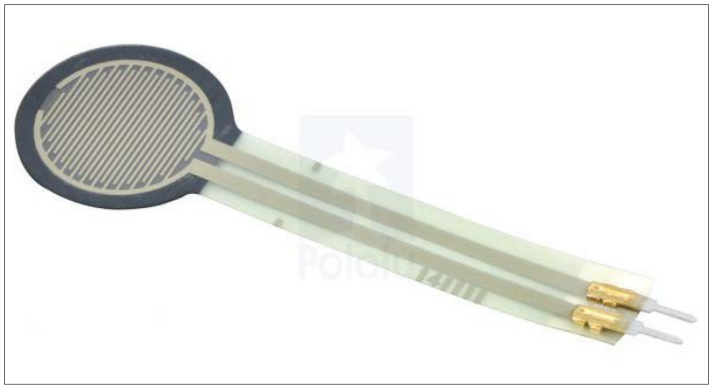
Using force sensors

Force sensors are really cool. They take the amount of force applied to them and return a signal. There are two different ones that I have used, and they are functionally the same. There is the one by Phidgets, in which we can easily make use of the code that we wrote for the light sensor. We can just attach a force sensor, which can than interact with a servo. We don't even need to change any part of the code.



The other option is a force sensor that is connected directly to the circuit. You can use it if you want to avoid a solution such as Phidgets. These are generally less expensive (around $7 USD).

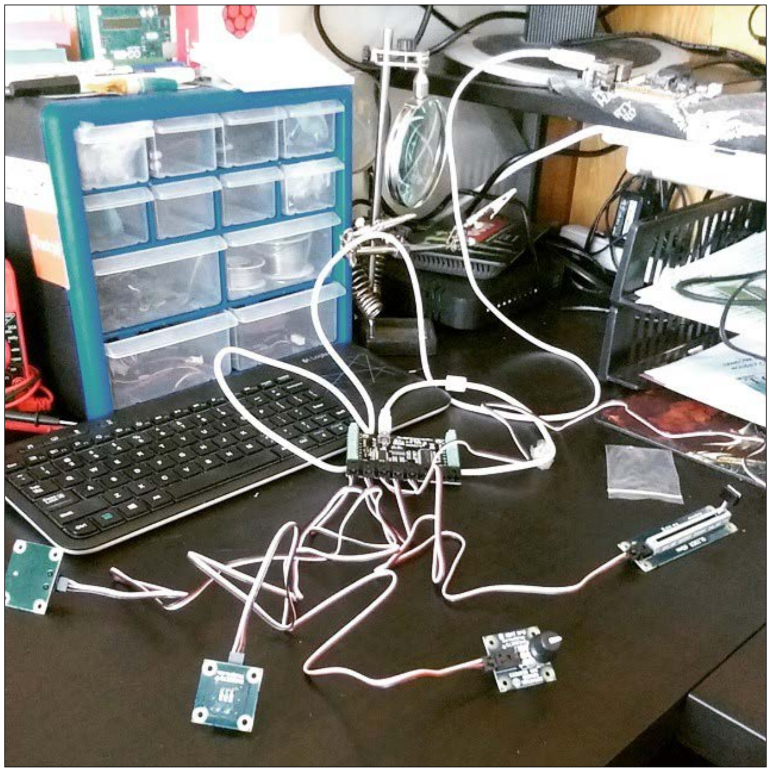
The sensor that I use looks like a circle with a rectangle coming out of it, as shown in the following image:



It is not a complicated component to add to your circuit. You can connect it to your breadboard and as long as you connect it to the GPIO pins, you can read the output from the sensor. There are only two pins. So, if you adapt a previous script, you can easily read the sensor.

Summary

In this chapter, we explored a lot of different sensors that allow us to develop a lot of different projects. One thing to remember is that as you add more sensors to your project, it can get pretty messy. You can see the aftermath of this in the following image:



In the preceding image, I have my control board connected to a touch sensor, slider, rotation sensor, and a light sensor. You can see the mess that comes with all the functionality. This can be managed.

In the next chapter, we will cover some exciting projects. The goal will be to take everything that we learned in this book so far and build an Internet radio. This will take advantage of several sensors, the Banana Pi, an IR receiver, and an exciting device—an LCD display. The Internet radio that I built for myself is inside an antique radio and is shown in the following image:

We will kick it up a notch from what I built and make something more modern.

7
Building an Internet Radio

So far, we have covered a lot of different topics. You learned about the different parts of the Banana Pi. You also learned how to use it to build awesome stuff. You have an understanding of the Linux operating system, and you are ready to make some amazing things.

In this chapter, we will explore a couple of new hardware components and build an Internet radio player. We will be able to use this player to listen to streams on the Internet, such as podcasts and music. This is one of the most popular projects. There are different approaches to this project, which also make it fun.

We will implement an LCD display and set up the software and some other hardware components. They will help us change channels and view the current stream. We will also take a look at the Banana Pi attachments and implement the same project using an attachment called the PiFace command and control. There are two ways to do the same project. The following image is of an Internet radio:

Setting up the music

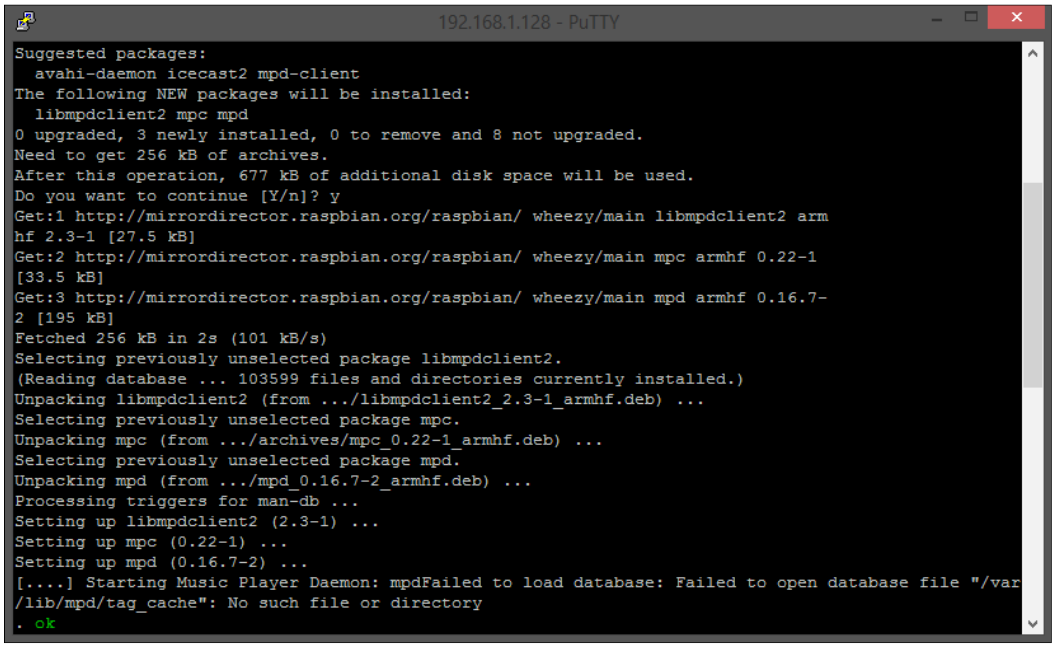
Arguably, the most important part of an Internet radio is its ability to play music. We will install MPC and MPD and add some radio stations. This isn't a complex configuration. We will start by updating packages.

```
sudo apt-get update
```

Now, we will install the packages we will use:

```
sudo apt-get install mpc
sudo apt-get install mpd
```

The following screenshot shows the output generated by installing MPC and MPD:



```
Suggested packages:
  avahi-daemon icecast2 mpd-client
The following NEW packages will be installed:
  libmpdclient2 mpc mpd
0 upgraded, 3 newly installed, 0 to remove and 8 not upgraded.
Need to get 256 kB of archives.
After this operation, 677 kB of additional disk space will be used.
Do you want to continue [Y/n]? y
Get:1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main libmpdclient2 arm
hf 2.3-1 [27.5 kB]
Get:2 http://mirrordirector.raspbian.org/raspbian/ wheezy/main mpc armhf 0.22-1
[33.5 kB]
Get:3 http://mirrordirector.raspbian.org/raspbian/ wheezy/main mpd armhf 0.16.7-
2 [195 kB]
Fetched 256 kB in 2s (101 kB/s)
Selecting previously unselected package libmpdclient2.
(Reading database ... 103599 files and directories currently installed.)
Unpacking libmpdclient2 (from .../libmpdclient2_2.3-1_armhf.deb) ...
Selecting previously unselected package mpc.
Unpacking mpc (from .../archives/mpc_0.22-1_armhf.deb) ...
Selecting previously unselected package mpd.
Unpacking mpd (from .../mpd_0.16.7-2_armhf.deb) ...
Processing triggers for man-db ...
Setting up libmpdclient2 (2.3-1) ...
Setting up mpc (0.22-1) ...
Setting up mpd (0.16.7-2) ...
[....] Starting Music Player Daemon: mpdFailed to load database: Failed to open database file "/var
/lib/mpd/tag_cache": No such file or directory
. ok
```

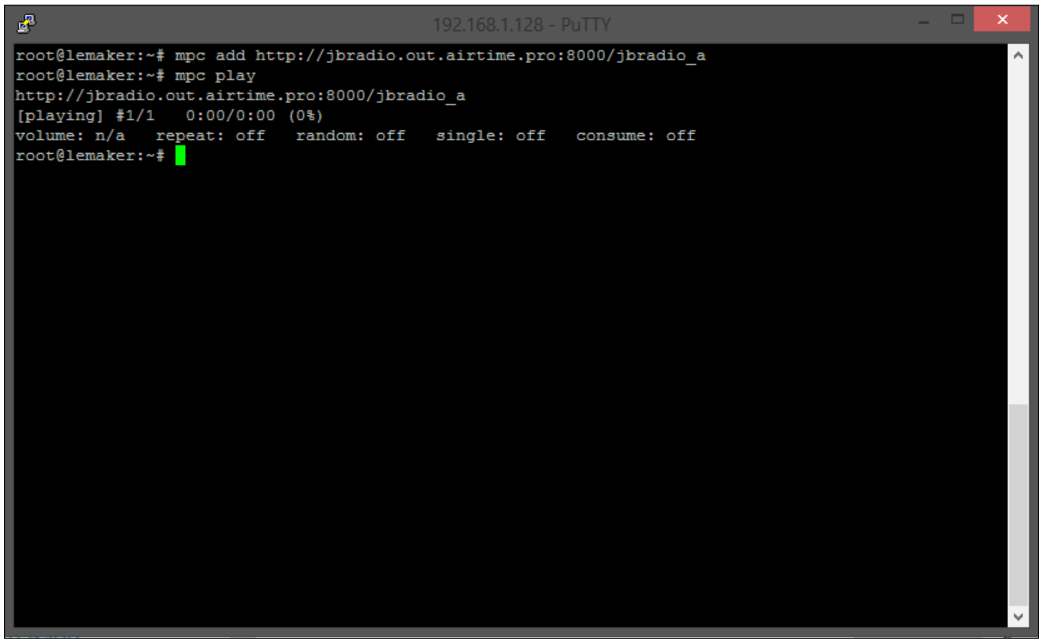
We can now add the radio stations we want to listen to. I will add my favorite podcast network, Jupiter Broadcasting, but you can add any audio stream you want. Run the following command multiple times with different URLs to add various channels to your playlist:

```
mpc add http://jbradio.out.airtime.pro:8000/jbradio_a
```

We can test this to make sure it is working by using the following command. Connect headphones or speakers.

```
mpc play
```

The following screenshot shows the MPC software playing a steam that was added:



```
root@lemaker:~# mpc add http://jbradio.out.airtime.pro:8000/jbradio_a
root@lemaker:~# mpc play
http://jbradio.out.airtime.pro:8000/jbradio_a
[playing] #1/1   0:00/0:00 (0%)
volume: n/a   repeat: off   random: off   single: off   consume: off
root@lemaker:~#
```

If you are happy with the playlist, you can save it using the following command:

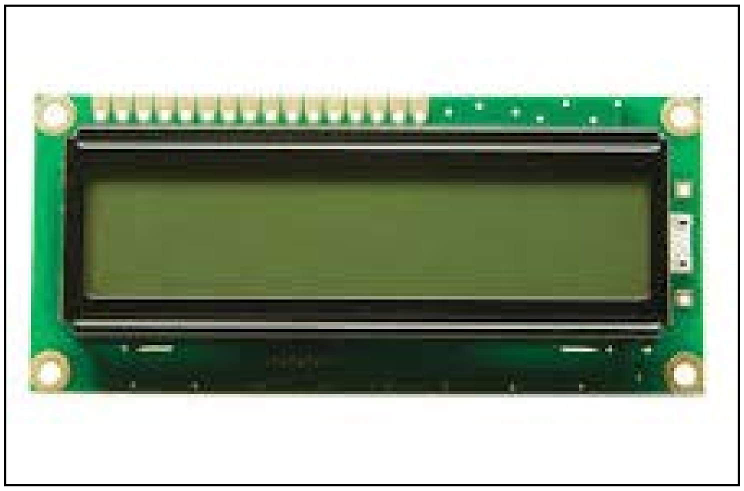
```
mpc save playlist
```

The playlist will be saved to the `/var/lib/mpd/playlists/` directory. You can edit the playlist manually using nano, or you could load the playlist into MPC as an argument if you want to listen to it again. You can also supply just the name of the playlist as an argument to start playing.

Using LCD displays

LCD displays are one of the best ways to display output from the Pi. It can be difficult to get started with using LCD displays, and there are innumerable choices. If you look at an option such as the PiFace or Adafruit's display boards, they connect directly to the GPIO pins.

The following image shows a basic LCD:



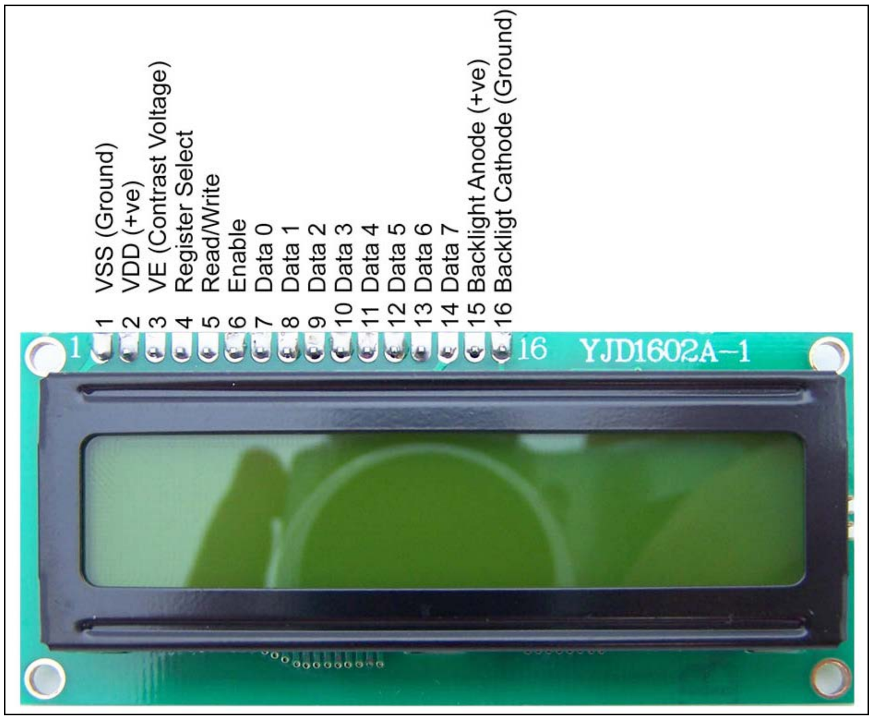
We will work with a standard 16 x 2 LCD display. We need to wire up the display to the Banana Pi.

Wiring up the LCD

Since we are directly connecting the LCD to the Banana Pi, we will wire it up. We won't have a GPIO header to connect directly to. Displays of size 16 x 2 generally have 16 pins. These devices are generally easy to work with. The different pins can be broken down into the following types:

- Data pins: These are the pins that send data to the display. These pins are pretty straightforward.
- Register Select pin: This can be used to move the display up or down or even clear it.
- Read/Write pin: This pin allows you to read and write to the LCD. This will almost always be in write mode. However, on rare occasions, it will be in read mode to allow us to read from the display.
- Enable pin: This is a simple pin that allows us to write to the registers on the LCD.

The following image shows an HD44780 LCD (16 x 2) pins labeled:



Pin definitions for a standard 16 x 2 LCD display can be seen in the following table:

Pin	Definition	Symbol
1	GND	VSS
2	+5V	VDD
3	Contrast adjustment	V0
4	H/L Register Select signal	RS
5	H/L Read/Write signal	R/W
6	H/L Enable signal	E
7	H/L Data bus line	DB0
8	H/L Data bus line	DB1
9	H/L Data bus line	DB2

Pin	Definition	Symbol
10	H/L Data bus line	DB3
11	H/L Data bus line	DB4
12	H/L Data bus line	DB5
13	H/L Data bus line	DB6
14	H/L Data bus line	DB7
15	+4.2V	A
16	Power supply for back light	K

Since we are working with the Banana Pi, it is important to verify that the LCD display you are using has an LED backlight as opposed to an EL backlight. EL backlights use a lot more power and are cheap to get, but will not pair well with any Pi. Displays with LED backlights have built-in resistors. If they don't, the Pi could be damaged. Most modern displays use LED backlighting now.

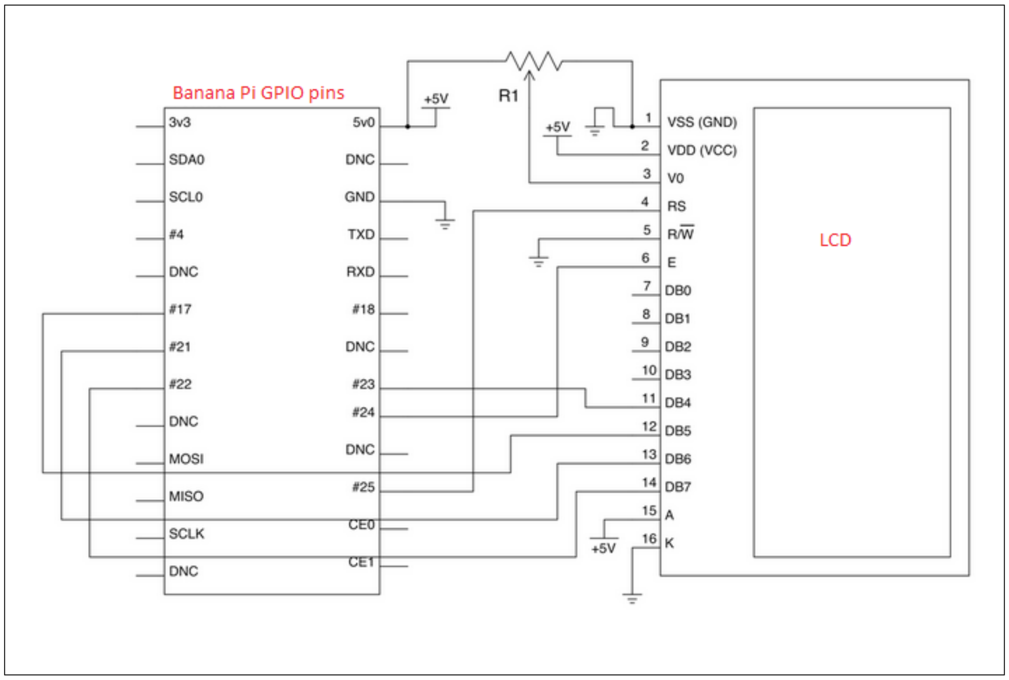
Connecting the Pi

Now that we have identified the pins on the LCD, we can match them up with GPIO pins. We will need to breadboard this at first (you can make a PCB later on). We will start by connecting the +5V (pin 2) power to the positive line on the breadboard and the GND pin (pin 1) to the black or negative line.

Now we can wire up the display to receive data from the Banana Pi. We will need a potentiometer (knob!) on the circuit as well. We will connect pin 3 to the potentiometer's middle prong. Pin 4 will connect to the negative line on the breadboard. Pin 6 will connect to pin 24 on the Banana Pi.

We will skip several pins. They are not necessary for what we are doing right now. These pins are 7, 8, 9, and 10; they are all data pins. Pin 11 connects to GPIO 23. Pin 12 connects to GPIO 17, and pin 13 connects to GPIO 21. Pin 14 connects to GPIO 22. Finally, pin 15 will connect to the red line, or positive rail, and pin 16 will connect to the ground.

Here is the schematic of the wiring diagram:



It is important to not send 5V from the LCD display into the Banana Pi, which is 3.3V. You need to make sure that there is proper resistance in place or use something like a Pi Cobbler from Adafruit when you breakout to the breadboard. You can hurt your Pi if you don't.

Writing to the display

We have wired up our display now. We want to write some text to the screen. We will, of course, need to make sure that the proper packages are installed. We installed the Python developer packages earlier, but just in case you skipped ahead, you will need to install them:

```
sudo apt-get install python-dev
sudo apt-get install python-setuptools
sudo apt-get install python-pip
```

We will also need the RPI GPIO library we used earlier in the book. You can conveniently find this library at `https://github.com/LeMaker/RPi.GPIO_BP`.

A single command will download the package from GitHub onto the Banana Pi:

```
git clone https://github.com/LeMaker/RPi.GPIO_BP -b bananapi
```

You can easily install this library by running the following few commands:

```
sudo apt-get update
sudo apt-get install python-dev
cd ./RPi.GPIO_BP
sudo python setup.py install
```

Now we will grab a script from Adafruit's GitHub. They wrote this script for the Raspberry Pi and to work with LCD displays, but it will work just fine for us:

```
git clone git://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code.git
```

Now we can use this script to test and change directories to the project and run the scripts to see the output. Now we want to take the music playing from the radio and display information on the LCD.

Radio output

We will start by pulling another repo from GitHub. This repo is by *Bob Rathbone* and includes some good scripts, such as `ada_radio.py`, which will allow us to output the music to the display:

```
git clone git@github.com:bobrathbone/piradio.git
```

We can use these scripts to bridge the gap between our LCD display and MPD. There are a few useful programs we will use. This will be even more useful if you are using a display such as the Adafruit complete display.

- `ada_radio.py`: This manages the LCD display
- `ada_lcd_class.py`: This class interfaces the LCD and buttons
- `I2C_class.py`: This is an i2c interfacing class
- `Test_ada_lcd.py`: This is a service to test the LCD functions

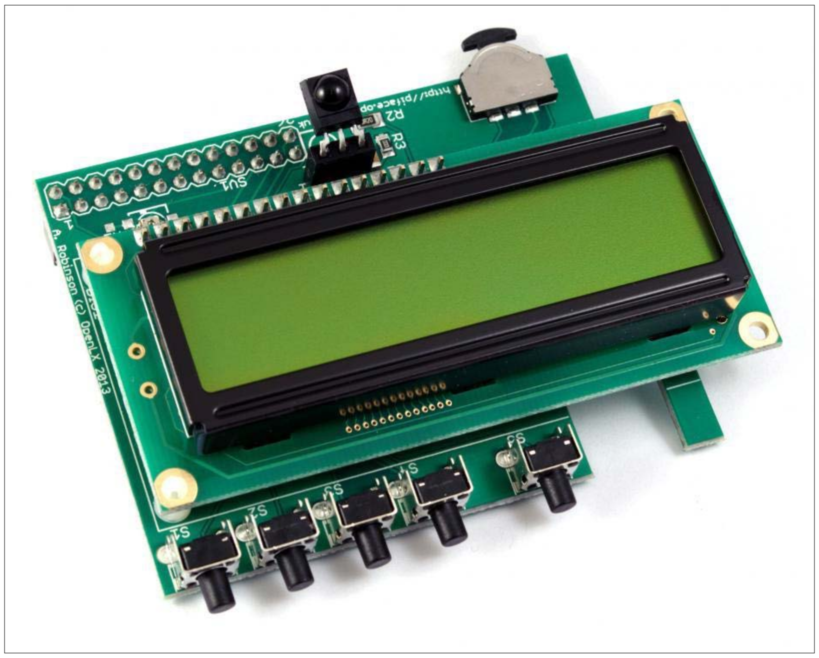
Although these tools are developed by Adafruit, they are useful in other situations as well. We will start the `ada_radio` script now:

```
python ada_radio.py start
```

This command will start the script, and the display should start to show on the LCD. The content displayed is coming from the MPD library we installed earlier. The information from that program is being logged to the display.

Introducing the PiFace radio

So, we spent some time manually building an Internet radio. This is awesome, but when you start to look at alternative LCD displays, such as the PiFace, you start to realize that there are some easier solutions. The following image shows the PiFace command and control:



We talked about this alternative earlier in the book, so I won't go into the details again. We will jump right into working with it.

As I'm sure you will have already guessed, we will start by installing the PiFace packages:

```
sudo apt-get install python3-pifacecad
```

Once they are installed, we will need to reboot:

```
sudo reboot
```

After the Banana Pi has rebooted, we can try out the display to make sure it actually works. We will change directories to the examples folder:

```
cd /usr/share/doc/python3-pifacecad/examples
```

We will need to make sure that we have the SPI module enabled under `raspi-config`. You may get an error like this:

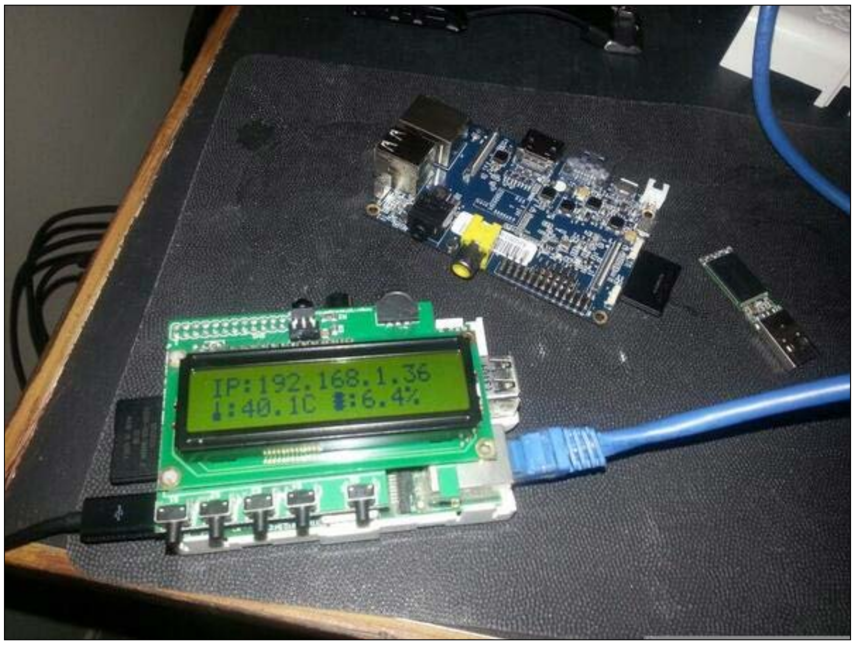
```
Traceback (most recent call last):
  File "/usr/lib/python3/dist-packages/pifacecommon/spi.py", line 40, in
open_fd
    self.fd = posix.open(spi_device, posix.O_RDWR)
OSError: [Errno 2] No such file or directory: '/dev/spidev0.1'


During handling of the above exception, another exception occurred:


Traceback (most recent call last):
  File "sysinfo.py", line 61, in <module>
    cad = pifacecad.PiFaceCAD()
  File "/usr/lib/python3/dist-packages/pifacecad/core.py", line 40, in
__init__
    super(PiFaceCAD, self).__init__(hardware_addr, bus, chip_select)
  File "/usr/lib/python3/dist-packages/pifacecommon/mcp23s17.py", line
91, in __                                          init__
    super(MCP23S17, self).__init__(bus, chip_select)
  File "/usr/lib/python3/dist-packages/pifacecommon/spi.py", line 32, in
__init_                                          _
    self.open_fd(spi_device)
  File "/usr/lib/python3/dist-packages/pifacecommon/spi.py", line 44, in
open_fd
    % (spi_device, SPI_HELP_LINK)
pifacecommon.spi.SPIInitError: I can't see /dev/spidev0.1. Have you
enabled the                                          SPI module?
(http://piface.github.io/pifacecommon/installation.html#enable-the-s
pi-module)
```

By opening the `raspi-config` menu in the terminal, we can go to the advanced options and enable the module. It will require a reboot; after the reboot, the error should be cleared up. If that doesn't work, you may need to reinstall the Python packages or the PiFace common package.

We can run the `sysinfo` example, which will enable the display and show us our local IP address and temperature:

```
python3 sysinfo.py
```

The output of the PiFace will look like this:



Now that we have this working, you can stop it by pressing *Ctrl* + *C*. We can now set up the Internet radio side of things.

The PiFace radio

We need to install mplayer on the Pi to set up the radio. We will run the following command and press *Y* whenever we are asked:

```
sudo apt-get install mplayer
```

We can now unzip the radio examples, which are in the same directory as the `sysinfo` example:

```
sudo gunzip /usr/share/doc/python3-pifacecad/examples/radio.py.gz
```

We can now run the PiFace radio:

```
python3 /usr/share/doc/python3-pifacecad/examples/radio.py
```

We have multiple controls on the PiFace. We can use these to change the radio stations and pause the stream.

Adding stations

Now we have a radio with some fancy buttons, but we have some stations we don't like. We can easily edit the software to add the stations we like. We will first need to copy the program from the examples directory to our home directory:

```
cp /usr/share/doc/python3-pifacecad/examples/radio.py ~
cp /usr/share/doc/python3-pifacecad/examples/radiolircrc ~
```

We will need to edit the radio `.py` file:

```
nano radio.py
```

We will look for the section of the code that identifies the stations as an array:

```
{'name': "station name",
'source': 'station web address',
'info': 'station info web address' or None},
```

I'll add the station I added in the first radio:

```
{'name': "Jupiter Broadcasting",
'source': http://jbradio.out.airtime.pro:8000/jbradio_a ',
'info': None},
```

Now, just add as many stations as you like. When you run the program again, you will be able to switch between these stations.

Making our radio pretty

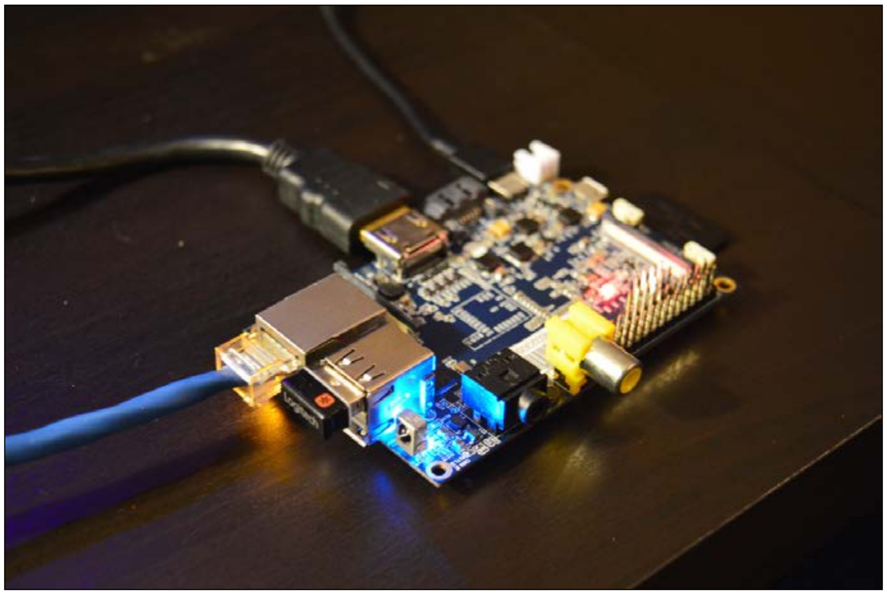
A Banana Pi radio is a quick and easy thing to set up. It is one of the best first projects that you can create because you get to experience both hardware and software, and you are making a functional device. Now that we have a functional device (well, actually two), you should be inspired to make something even more awesome.

I took one of my Banana Pis and thought an Internet radio was cool. However, it would be even cooler if it was in an antique radio. Here is what it ended up looking like:

I hollowed out the inside of this antique radio. It wasn't working anymore, so don't worry about that part. I also broke apart some spare speakers I had lying around. The original volume knob is glued to the potentiometer of the speaker on the inside, giving it the proper functionality.

Everything else we talked about in this chapter boils down to the same end goal: a project that allows you to be creative. The case that I chose unfortunately didn't make sense adding an LCD. However, I used all the other aspects of this chapter, and it created a very functional device to listen to my podcasts.

You could always leave the Banana Pi out of its case, like this one:

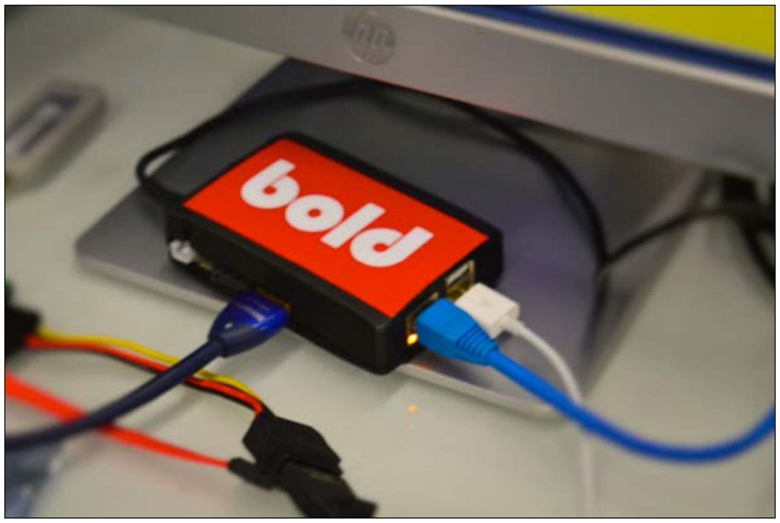


There are all kinds of containers for your Internet radio. You don't need to look for antique radios; however, they do make for a nice project. You could also mount the Banana Pi behind your monitor and connect it to speakers that you control via SSH. Alternatively, you could put the Banana Pi with your stereo system and control it by a web interface. Because of the open nature of the product, you can do almost anything you want.

Some enclosures that you could use for this radio project or even for other situations could be fabricated from a lot of interesting things, such as the following:

- LEGO
- N64 cartridge
- Mint tin

If you have access to a 3D printer like I do, you could even 3D-print a case such as the one that I made:



Summary

The Internet radio we made is a very practical tool. Everybody likes listening to music or podcasts, or really any media. We have taken that love for music and built something out of it. This project is a jumping-off point for future projects. You can take what you have learned here and apply it to something such as an ownCloud server or media PC.

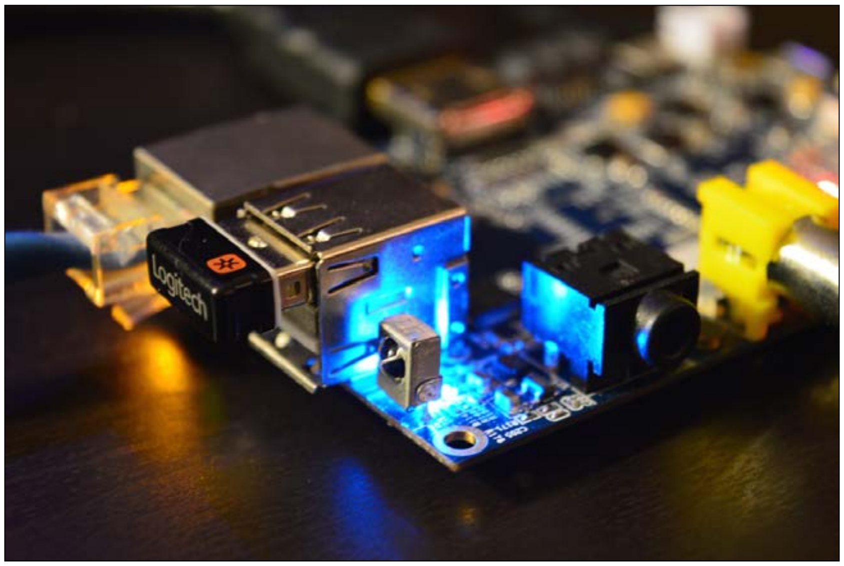
A few potential projects to get you inspired to continue learning include:

- HTPC
- Workstation
- Retro Gaming Emulator
- Internet Kiosk
- Wireless Access Point
- Portable web server

These projects are generally pretty easy to implement and make for good weekend projects.

We have taken all the accumulated knowledge from this book so far and applied it to a practical project. We went from knowing almost nothing about single-board computing to understanding the hardware aspects of the board and learning about the Linux filesystem and basic commands. We programmed a lot of different stuff directly on the board using Ruby, Go, Python, and Bash. We applied all this knowledge to make this radio.

In the next chapter, we will cover some other practical applications, including running a web server on the Banana Pi for use with server-side programming with PHP. We will install a local WordPress installation. We will take this tiny server (look at the following image) and turn it into a powerful web server for local development:



8
Building a Home Server

Now that we have explored a lot of hardware and some programming, we can start talking about another popular use of the Banana Pi: home servers. Earlier, to get a home server, you needed to take one of your existing computers that you had kicking around and turn it into a server. These machines were usually older PCs or expensive dedicated hardware. Now we can accomplish this task with a simple device. The cost of running a Banana Pi is much less than the cost of running a PC since we are powering the device with USB.

There are a few different kinds of servers we can run with a Banana Pi. Some of the servers we will explore include a web server for local development and web pages (dashboard style), ownCloud servers to run local cloud DropBox alternatives, and NAS (network access storage) servers. The following image shows my Banana Pi server:

Setting up a web server

Web servers serve web pages. A web server might seem like something silly to have in your house, but there are a lot of practical applications. Of course, if you are a web developer, it becomes even more useful. However, even if you're not a developer, you can set up something such as a dashboard that shows the weather.

We will start by installing the base software that will be necessary, regardless of what you want to do here. We will install nginx to serve web pages, MySQL as our database, and PHP5, which we will use for the web pages.

Setting up our server

We will start by installing nginx using the following command:

```
sudo apt-get install nginx
```

Now, to start nginx, we will run the following command:

```
/etc/init.d/nginx start
```

You can test nginx by going to the IP address of your Banana Pi in the web browser on the Pi itself or on another computer.

Setting up MySQL is easy, and the installation process will guide you through the steps. Here's the command to install MySQL:

```
sudo apt-get install mysql-server
```

Now you will be prompted for the root password. Set this as something you can remember, because it is important. Most of the defaults will be fine for what we are here for.

Installing PHP will require a few different packages. We will install the packages to get PHP working with MySQL. We will speed it up a bit because we are running on a single-board computer after all. We will need to run a few commands, which are as follows:

```
sudo apt-get install php5-fpm
sudo apt-get install php5-mysql
sudo apt-get install php-arc
```

Now we need to configure nginx. PHP is already configured to work with nginx. However, we need to comment out a few lines in the configuration file `/etc/nginx/sites-available/default`:

```
    listen   80; ## listen for ipv4; this line is default and
    implied
    #listen   [::]:80 default_server ipv6only=on; ## listen for ipv6

  < more content >

    location ~ \.php$ {
      fastcgi_split_path_info ^(.+\.php)(/.+)$;

      # php5-cgi alone:
    #  fastcgi_pass 127.0.0.1:9000;
      # php5-fpm:
      try_files $uri $uri/ /index.php?$uri&$args;
      fastcgi_pass unix:/var/run/php5-fpm.sock;
      fastcgi_index index.php;
      include fastcgi_params;
    }
```

We need to make sure we add the following line:

```
try_files $uri $uri/ /index.php?q=$uri&$args;
```

We also need to comment out the following line, because we are using `php-fpm`:

```
fastcgi_pass 127.0.0.1:9000;
```

We also need to edit the `php.ini` file, which contains all the configuration options for PHP on the server. We need to set `cgi.fix_pathinfo` to 0.

We will then edit the `php.ini` file:

```
nano /etc/php5/fpm/php.ini
```

Now we will set the parameter. This file is pretty big, but you can press *Ctrl* + *W* to search the document. When you find the parameter, you will need to set it:

```
cgi.fix_pathinfo=0
```

Now we will need to reset our server since we have made changes to the configuration files. On start in the future, the server will load with our changes.

```
/etc/init.d/nginx restart
```

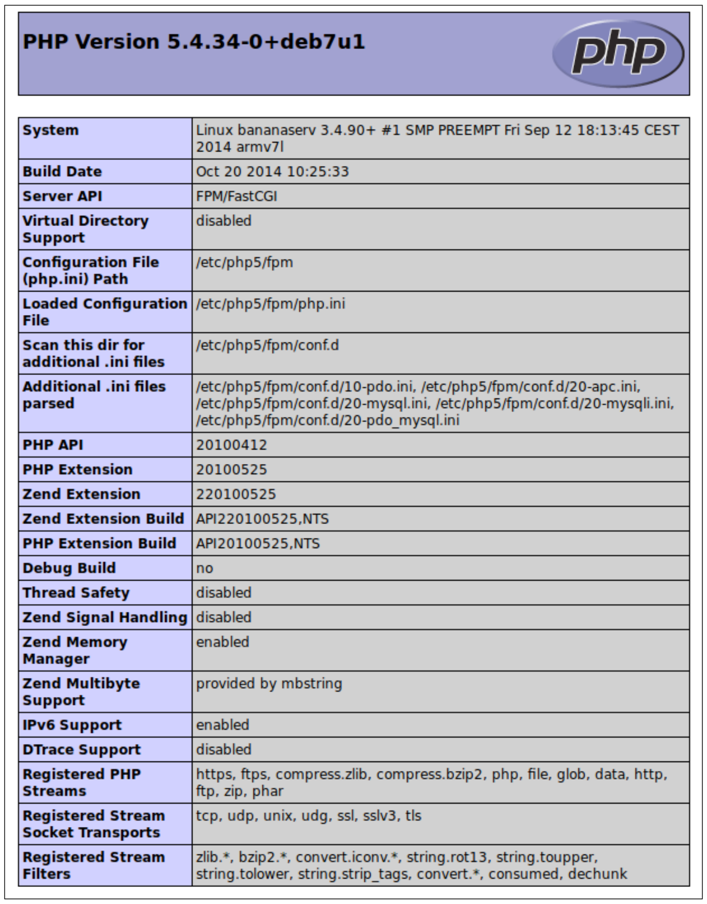
We can create a simple page to check whether the server is working properly with PHP. We need to create a file called `info.php`. This file will output all the server information:

```
nano /usr/share/nginx/www/info.php
```

We need to put some simple code in this file for it to work:

```
<?php
phpinfo();
?>
```

The page can be viewed by going to your Banana Pi's IP address in the browser and adding the page information, `/info.php`, to the URL. If your Banana Pi is on IP address `192.168.1.25`, we use `http://192.168.1.25/info.php`. If everything is working properly, you would see a page that looks like this:



PHP Version 5.4.34-0+deb7u1

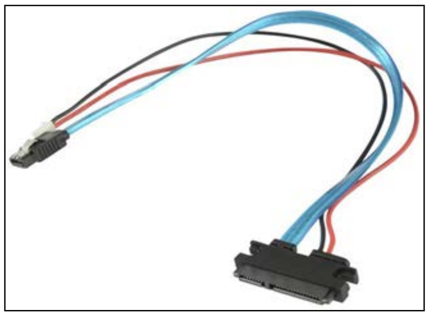
System	Linux bananaserv 3.4.90+ #1 SMP PREEMPT Fri Sep 12 18:13:45 CEST 2014 armv7l
Build Date	Oct 20 2014 10:25:33
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/fpm
Loaded Configuration File	/etc/php5/fpm/php.ini
Scan this dir for additional .ini files	/etc/php5/fpm/conf.d
Additional .ini files parsed	/etc/php5/fpm/conf.d/10-pdo.ini, /etc/php5/fpm/conf.d/20-apc.ini, /etc/php5/fpm/conf.d/20-mysql.ini, /etc/php5/fpm/conf.d/20-mysqli.ini, /etc/php5/fpm/conf.d/20-pdo_mysql.ini
PHP API	20100412
PHP Extension	20100525
Zend Extension	220100525
Zend Extension Build	API220100525,NTS
PHP Extension Build	API20100525,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	https, ftps, compress.zlib, compress.bzip2, php, file, glob, data, http, ftp, zip, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, tls
Registered Stream Filters	zlib.*, bzip2.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk

Setting up ownCloud

ownCloud is a DropBox alternative. It allows you to create your own service, such as DropBox, but you get to own the hardware. The image at the beginning of this chapter of my Banana Pi server is actually my ownCloud server. We need to do a couple of things to get this working. If we are setting up a cloud service, we will likely want more storage than we will get from our SD card. Luckily for us, we have access to a SATA port on the board.

Setting up the SATA drive

So, we have a SATA drive and we want to mount it to a spot where we can access and store data to the drive. For this purpose, we will assume that the drive has already been formatted, and we will start mounting the drive. In order to connect a drive to the Banana Pi and power it, you need a fancy cable like the one shown here:



You need a cable because on the Banana Pi, there is a power connector that can power a hard drive. You can use an external power source, but this one will power the drive while the Banana Pi is powered on. You will connect the power adapter to the pins on the board and then connect the SATA connector to the board. The other end of the cable is one large connector to connect directly to the board.

Once the drive is connected and the Banana Pi is powered on, the drive will come to life. The devices connected to the device can be found by running a quick command:

```
fdisk -l
```

You will be presented with output that identifies all the disks attached to the device. My disk was `/dev/sda1/`. We can mount this folder to a directory. We will create a directory in `/media/`. This directory will hold our data:

```
mkdir /media/data/
```

Now we will mount the SATA drive to our new directory:

```
mount /sev/sdb1/ /media/data
```

When we look in that directory now, it will be an empty folder. We can now use this folder to hold the files added by ownCloud.

Installing ownCloud

We have a Banana Pi with a web server and a SATA drive connected and mounted. We can now install ownCloud. Since we are installing the ownCloud server onto our nginx server, we need to set up SSL so that our user session isn't hijacked. We will run a few commands:

```
mkdir /etc/ssl
mkdir /etc/ssl/nginx
cd /etc/ssl/nginx
openssl genrsa -out privkey.pem 2048
openssl req -new -x509 -key privkey.pem -out cacert.pem -days 1095
```

We are creating directories to hold the SSL certificates and generating our keys in order to decrypt our traffic. Now we need to set up our database connection. We need to log in to MySQL:

```
mysql -u root -p
```

You will be prompted for your root password you set up earlier (I hope you remember it). We need to create a user and a database for the ownCloud installation:

```
CREATE USER 'cloud'@'localhost' IDENTIFIED BY 'your_password';
CREATE DATABASE IF NOT EXISTS owncloud;
GRANT ALL PRIVILEGES ON owncloud.* TO 'cloud'@'localhost' IDENTIFIED BY
'your_password';
FLUSH PRIVILEGES;
```

We can now log in as the user cloud and whatever password you set will be used to connect to it.

You can now quit MySQL easily:

```
quit
```

We now need to install a few extra packages that will be needed for ownCloud:

```
apt-get install php5-gd php5-json php5-curl php5-intl php5-mcrypt php-
xml-parser
```

These packages basically extend the functionality of PHP. We will also need to install bzip:

```
apt-get install bzip2
```

We are now ready to install ownCloud:

```
wget https://download.owncloud.org/community/owncloud-7.0.3.tar.bz2
tar -jxvf ownCloud 7.0.3.tar.bz2
cp –rf  ./ownCloud/ /var/www
```

We need to change the owner of the files we just moved to the `/var/www` directory so that our web server can use them:

```
chown –R www-data:www-data /var/www
```

We have all the files we need to get going. Now we need to edit some configuration files. We will start by editing the php.ini file again, this time adjusting the file size limitations so that we can sync larger files:

```
sudo nano /etc/php5/fpm/php.ini
```

We will need to edit the following lines, which are not exactly together in the file. You need to find them:

```
upload_max_filesize = 1024M
post_max_size = 1024M
```

Now, we will configure nginx again, this time to make sure that the secure ownCloud install will be available over HTTPS:

```
sudo nano /etc/nginx/sites-available/ssl-default
```

We can use this configuration to get it running:

```
upstream php-handler {
    # server 127.0.0.1:9000;
    server unix:/var/run/php5-fpm.sock;
}
server {
  listen 443;
  ssl on;
  ssl_certificate /etc/ssl/nginx/cacert.pem;
  ssl_certificate_key /etc/ssl/nginx/privkey.pem;
  root /var/www;
  client_max_body_size 1G;
  fastcgi_buffers 64 4K;
  rewrite ^/caldav(.*)$ /remote.php/caldav$1 redirect;
  rewrite ^/carddav(.*)$ /remote.php/carddav$1 redirect;
  rewrite ^/webdav(.*)$ /remote.php/webdav$1 redirect;
  index index.php;
  error_page 403 /core/templates/403.php;
  error_page 404 /core/templates/404.php;
  location = /robots.txt {
    allow all;
    log_not_found off;
    access_log off;
  }
    location ~ ^/(data|config|\.ht|db_structure\.xml|README) {
      deny all;
    }
    location / {
    # The following 2 rules are only needed with webfinger
    rewrite ^/.well-known/host-meta /public.php?service=
    host-meta last;
    rewrite ^/.well-known/host-meta.json /public.php?service
    =host-meta-json last;
    rewrite ^/.well-known/carddav /remote.php/carddav/ redirect;
    rewrite ^/.well-known/caldav /remote.php/caldav/ redirect;
    rewrite ^(/core/doc/[^\/]+/)$ $1/index.html;
    try_files $uri $uri/ index.php;
    }
    location ~ ^(.+?\.php)(/.*)?$ {
        try_files $1 =404;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$1;
        fastcgi_param PATH_INFO $2;
        fastcgi_param HTTPS on;
        fastcgi_pass php-handler;
    }
}
```

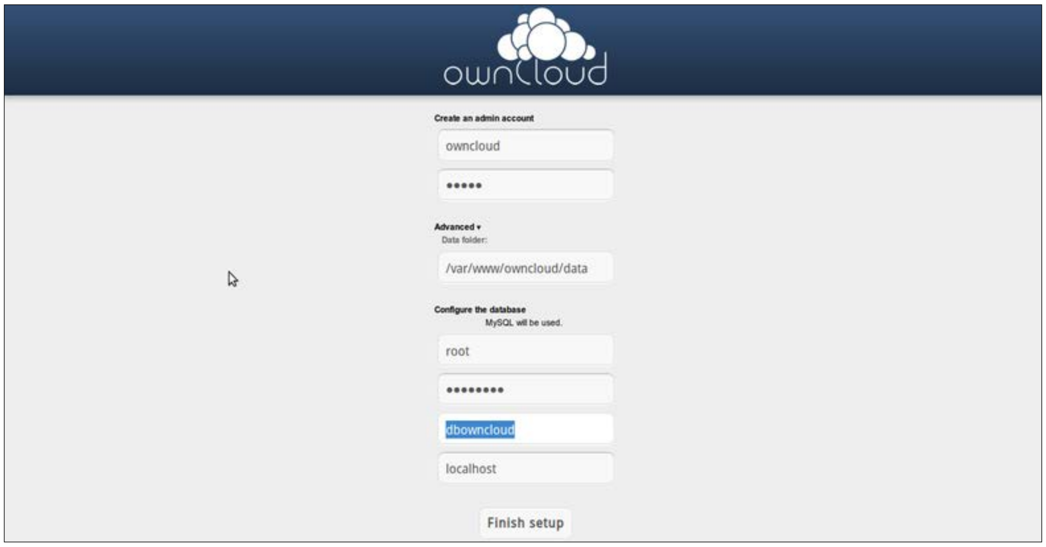
If you have deviated from the instructions, you will need to make sure you adjust this configuration accordingly. We will create a symlink to sites-enabled:

```
ln -s /etc/nginx/sites-available/ssl-default /etc/nginx/sites-enabled
```

Now we can restart our server:

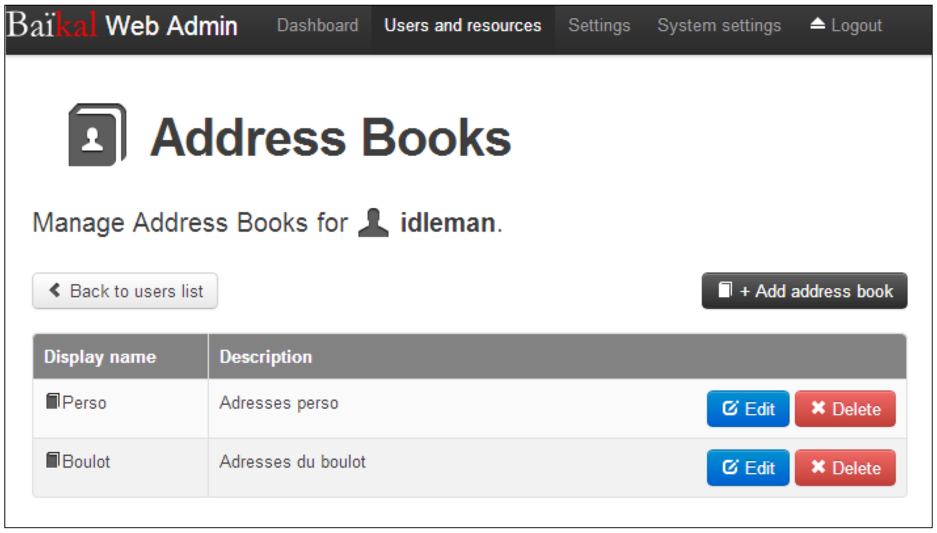
```
/etc/init.d/php5-fpm restart
/etc/init.d/nginx restart
```

You will be able to access the ownCloud setup page from the browser. If you connect to your IP and navigate to ownCloud, you will see the page. On the initial install page, you will be asked to connect to the database you created earlier and tell it to put the data in the `/media/data` directory we created when we mounted the SATA drive.



Installing Baikal

Baikal is a way for you to sync your calendar and contacts without submitting to a third-party solution such as Google. Of course, ownCloud offers the same solution. It is a lot lighter since it is not doing the file syncing. The interface looks like this:



Address Books

Manage Address Books for idleman.

Display name	Description
Perso	Adresses perso
Boulot	Adresses du boulot

We already have nginx and PHP installed. However, this time around, we will use SQLite as opposed to MySQL.

We will install the packages needed. They include SQLite and the libraries needed for PHP to interact with it:

```
sudo apt-get install sqlite php5 php5-fpm php5-sqlite
```

We will also generate an SSL certificate (if you did this for ownCloud, you're still good):

```
sudo mkdir /etc/nginx/ssl
cd /etc/nginx/ssl
openssl genrsa -out privkey.pem 2048
openssl req -new -x509 -key privkey.pem -out cacert.pem -days 1095
```

Now we need to create a virtual host for nginx to route our requests to:

```
nano /etc/nginx/sites-available/ssl
```

Then we need to edit this file. You can use the following configuration for it to work properly:

```
server {
  listen 443;

  ssl on;
  ssl_certificate /etc/nginx/ssl/cacert.pem;
  # path to your cacert.pem
  ssl_certificate_key /etc/nginx/ssl/privkey.pem;
  # path to your privkey.pem

  root /var/www/;
  index index.html index.htm index.php;

  server_name localhost;
  location / {
    try_files $uri $uri/ /index.html;
  }
  location ~ ^(.+\.php)(.*) {
    try_files $fastcgi_script_name =404;
    fastcgi_split_path_info ^(.+\.php)(.*)$;

    fastcgi_pass unix:/var/run/php5-fpm.sock;
    fastcgi_param  SCRIPT_FILENAME
    $document_root$fastcgi_script_name;
    fastcgi_param  PATH_INFO        $fastcgi_path_info;
    fastcgi_index index.php;
    include fastcgi_params;
  }

      charset utf-8;

  location ~ /(\.ht|Core|Specific) {
    deny all;
      return 404;
  }
}
```

We will activate the host file now by creating the symbolic link to the sites-enabled file:

```
ln -s /etc/nginx/sites-available/ssl /etc/nginx/sites-enabled
```

We are now ready to download Baikal.

Downloading Baikal

Now that our server is prepped and ready, we can download the Baikal files and install the service. We will need to run a few commands to complete this task:

```
cd /var/www
wget http://baikal-server.com/get/baikal-flat-0.2.7.zip
unzip baikal-flat-0.2.7.zip
mv baikal-flat/ baikal/
chown -R www-data:www-data baikal/
```

We are good to go. We can now enable the installation by creating the empty file `ENABLE_INSTALL`, which, in my opinion, is the worst way ever to accomplish this task.

```
touch /var/www/baikal/Specific/ENABLE_INSTALL
```

Now you can go through the installation and create the administrator account and users. Once the users are created, you will be able access the contact book and calendar for those users. You can subscribe to these services through your clients, just like you would subscribe to any other calendar or contact service.

Setting up swap

Swap memory is very common in Linux. It acts as memory that is used when the physical RAM on the device is used up. This is not as necessary on newer computers using Linux because, for example, the laptop I am writing this on is running Ubuntu, and I have 16 GB of RAM.

Our Banana Pis have 1 GB of RAM. This is more than enough to perform simple tasks or run simple servers. However, once we are running PHP and it gets more complex, we need some extra memory. It is recommended that you do not create a swap file on the SD card, because it will be a slower medium. Also, it isn't worth using the file as memory and will shorten the life span of your card.

We will create a swap file on the drive we have connected to over SATA. This is going to be much faster. We will create a 2 GB swap file, which will give us 1 GB of RAM and 2 GB of extra swap memory.

We will start by running the swapon command:

```
sudo swapon -s
```

This command will give you output that looks like this:

```
Filename                Type        Size    Used    Priority
```

You will notice that there are no swap files. This means you currently only have the device memory, which means you may run out and notice some lag. We will fix this. We can also verify the absence of a swap file by running the free command:

```
free -m
```

The output will look something like this, showing that there is no swap memory available:

```
             total       used       free     shared    buffers     cached
Mem:          3953        154       3799          0          8         83
-/+ buffers/cache:         62       3890
Swap:            0          0          0
```

Now we will create a swap file on the drive we mounted to `/media/data`. Assuming that you have not gone through the ownCloud section yet, it might make sense to put this file in another directory on that drive from the data mount point. You may create an `/ownCloud` directory inside data and keep the swap file at the root:

```
dd if=/dev/zero of=/media/data/swapfile bs=1k count=2048k
```

We now have a 2-GB file called swapfile in `/media/data/`. We can now make this file active using `swapon`:

```
chmod 600 /media/data/swapfile
chown root:root /media/data/swapfile
sudo mkswap /media/data/swapfile
sudo swapon /media/data/swapfile
```

Our swap file is now activated. We can check it out by running our `swapon` command again:

```
sudo swapon -s
```

Now our output looks like this:

```
Filename                Type        Size    Used    Priority
/media/data/swapfile               file        4194300 0       -1
```

We have a working swap file, which is awesome, except that we want it to be permanent. We will edit the `fstab` file (which sounds dangerous, but isn't):

```
sudo nano /etc/fstab
```

At the very bottom of the file, we will add the following command:

```
/media/data/swapfile   none    swap    sw    0   0
```

Now, we can save and close the swap file, and this file will be working even if you reboot. One thing to note is that if you are going to remove the SATA drive, you should remove this swap file. Otherwise, it will be looking for a drive that isn't there, and as you may imagine, this is less than optimal.

The swap file we just added will help in the event we run out of memory. Processes such as MySQL can use more memory, which means their queries will be much faster, and the device will become a little more usable. We now have a nice little server for our home.

Summary

We covered a lot about servers but barely scratched the surface. You learned about a couple of practical home server solutions here. They will allow you to sync your files, calendars, and contacts without letting anybody else have your data, which is key today.

We explored a lot in this book. We built physical hardware that interacts with code, we built servers that can simplify our lives, and you learned a lot along the way. There is so much more that you can accomplish with the Banana Pi. My intention is to arouse your interest and thus motivate you to learn some more.

One thing that makes these boards so awesome is the community forums. You can search for almost anything you are trying to do and find help in doing it. There are a lot more operating systems available for the Banana Pi that we didn't cover in this book. These operating systems are interesting to work with as well. Depending on what you are trying to do, you will find that there are a lot of different options to get there.

9
Gaming on Your Pi

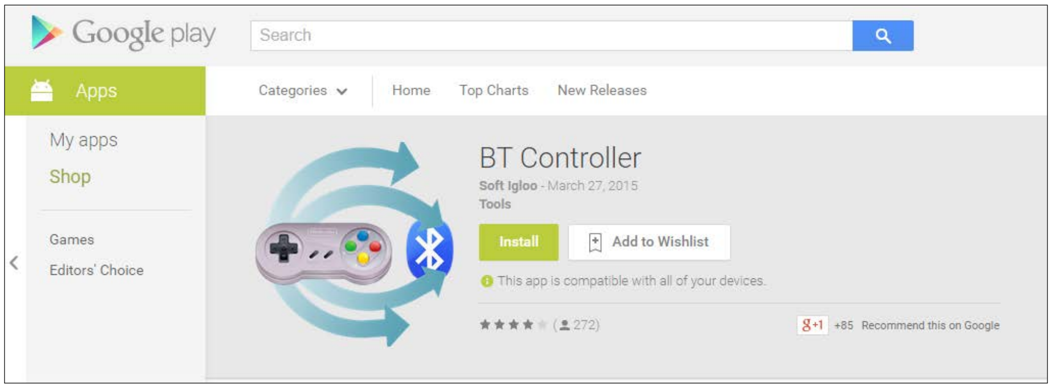
One of the cool things about the Banana Pi is that it allows you to basically do whatever you want. One of the applications of the Banana Pi is that it can be used as a media device, and even in this aspect, it isn't restricted to movies and music. The Banana Pi has a much more robust hardware than the Raspberry Pi. The Banana Pi allows you to play games on Android and Linux, which is something that the Raspberry Pi can't quite handle.

There are a few options when it comes to playing games on the Banana Pi. You are restricted to the fact you are still on Linux, which requires support. We are also running an Arm board. So, there will be a restriction on the hardware that the games are built for. I have found that retro game emulators and anything that is built for Android works great. The following picture shows Mario being played on the Banana Pi that runs on Android:

Gaming on Android

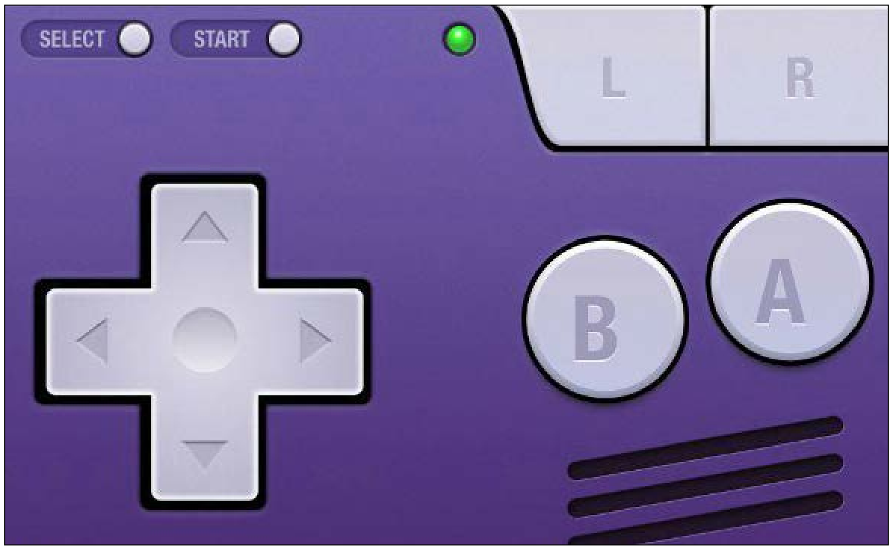
The main focus of this book has been running Linux and building some awesome stuff, but when it comes to gaming, Android beats Linux. Most of the games for Android are designed for a touchscreen. If you are using a touchscreen display, you are in luck. Otherwise, you will need to find some other alternatives.

Unfortunately, the Banana Pi lacks Xbox 360 controller support. However, other controllers are supported. The best controller that I have found so far is an app called BT Controller on the Google Play Store that I can use on my phone or tablet. Of course, this assumes that you are using an Android phone or tablet. The following screenshot shows the BT Controller app for Android:



Setting up BT Controller is simple. You have a couple of options that are easy to configure. First, on the Banana Pi, you will need to set the controller as the server. On the phone, you will need to search for the IP address of the Banana Pi. I have only been able to get the Wi-Fi part of this working as the Banana Pi has not been able to use any of my Bluetooth accessories on Android, but they work fine on Linux. This may be a hardware issue with the Bluetooth dongles that I have available here.

The following image shows the controller app running on a phone:



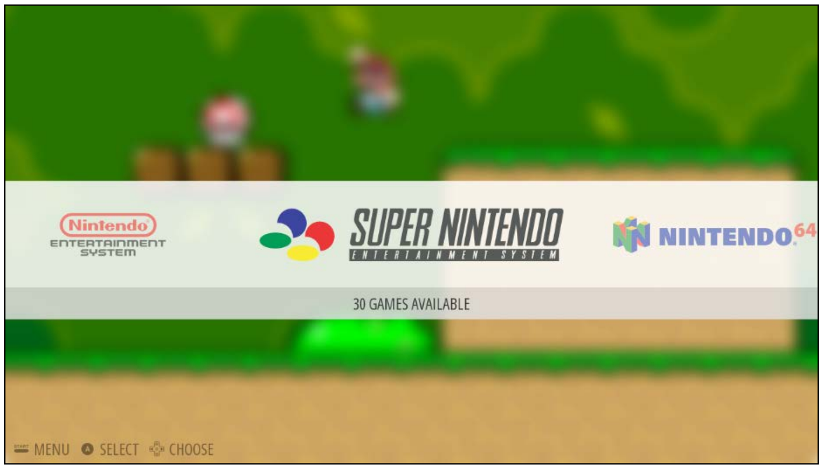
Some games work better than the others, but the hardware on the Banana Pi has been proven to provide great results. You are only going to miss out on touchscreen games, which won't be an issue if you have a touch-enabled display connected to the Banana Pi.

Classic game emulators for systems such as Nintendo 64 and Play Station work perfectly. I can't tell you where to get games for them. The controller support is generally quite robust and supports a perfect gameplay. At this point, the Banana Pi can be considered the ultimate media device.

Gaming on Linux

Gaming on Linux is something that has always been historically valid, but until recently, there wasn't a lot that you could do. Now, with the rise of some open source projects, there are lots of games that you can play on Linux that are much more professionally made, and some of these games are supported by the community.

You can do any number of searches for games that will run on the Raspberry Pi, Banana Pi, or Linux, and you will come across lists of the best titles that you can find. They won't say which particular games you can play, but will point you to some of the software you can use to play them.

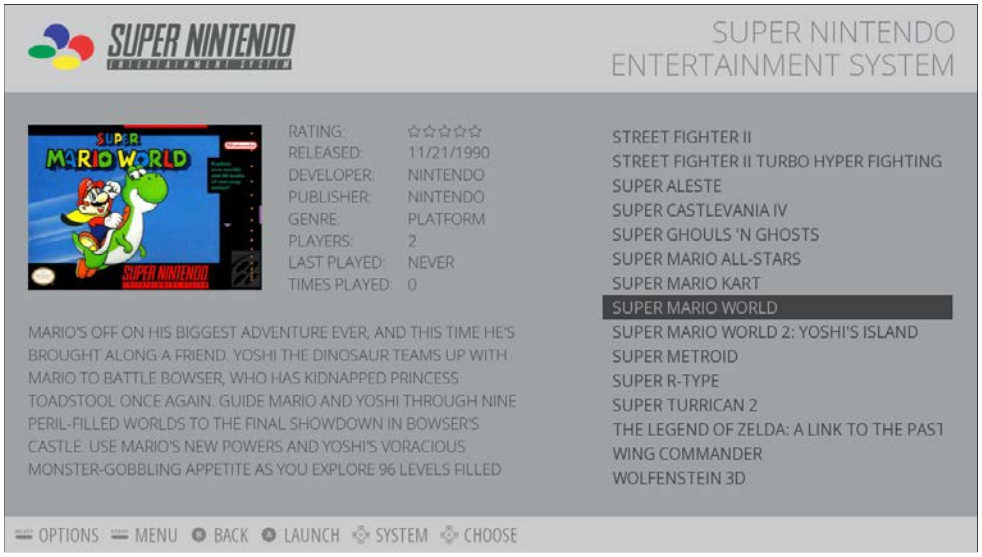

Nintendo ENTERTAINMENT SYSTEM
SUPER NINTENDO ENTERTAINMENT SYSTEM
NINTENDO 64
30 GAMES AVAILABLE
MENU
SELECT
CHOOSE

EmulationStation

Retro game emulation is one of the most popular uses of the Pi boards. It is a very cheap alternative to larger systems and can handle many different game emulators. There is an open source project called EmulationStation that handles many different systems.

There are a few different ways to install this and unfortunately, at the time of writing this book, there isn't a premade RetroPie image for the Banana Pi. So, we will have to install and compile the software manually.

The end result will be an easy-to-use emulation machine.



RATING:	☆☆☆☆☆
RELEASED:	11/21/1990
DEVELOPER:	NINTENDO
PUBLISHER:	NINTENDO
GENRE:	PLATFORM
PLAYERS:	2
LAST PLAYED:	NEVER
TIMES PLAYED:	0

MARIO'S OFF ON HIS BIGGEST ADVENTURE EVER, AND THIS TIME HE'S BROUGHT ALONG A FRIEND. YOSHI THE DINOSAUR TEAMS UP WITH MARIO TO BATTLE BOWSER, WHO HAS KIDNAPPED PRINCESS TOADSTOOL ONCE AGAIN. GUIDE MARIO AND YOSHI THROUGH NINE PERIL-FILLED WORLDS TO THE FINAL SHOWDOWN IN BOWSER'S CASTLE. USE MARIO'S NEW POWERS AND YOSHI'S VORACIOUS MONSTER-GOBBLING APPETITE AS YOU EXPLORE 96 LEVELS FILLED

Before we get started with the installation of the software, we will need to make sure that we have everything up to date. If you are using Raspbian, you should also update the firmware by using the following command:

```
sudo apt-get update
sudo apt-get upgrade
```

We can now install the EmulationStation software. We will need to edit the boot config file and add a line that will set the GPU memory to 32. We will do this so that we don't run out of memory while compiling the software.

```
sudo nano /boot/config.txt
```

We will add this line to the file, save, and exit, as follows:

```
gpu_mem = 32
```

Once we have saved and exited the file, we will immediately reboot the system, as follows:

```
reboot
```

After the system restarts, we will need to start installing the dependencies for SDL2. This is important. So, you will need each of these packages:

```
sudo rpi-update
```

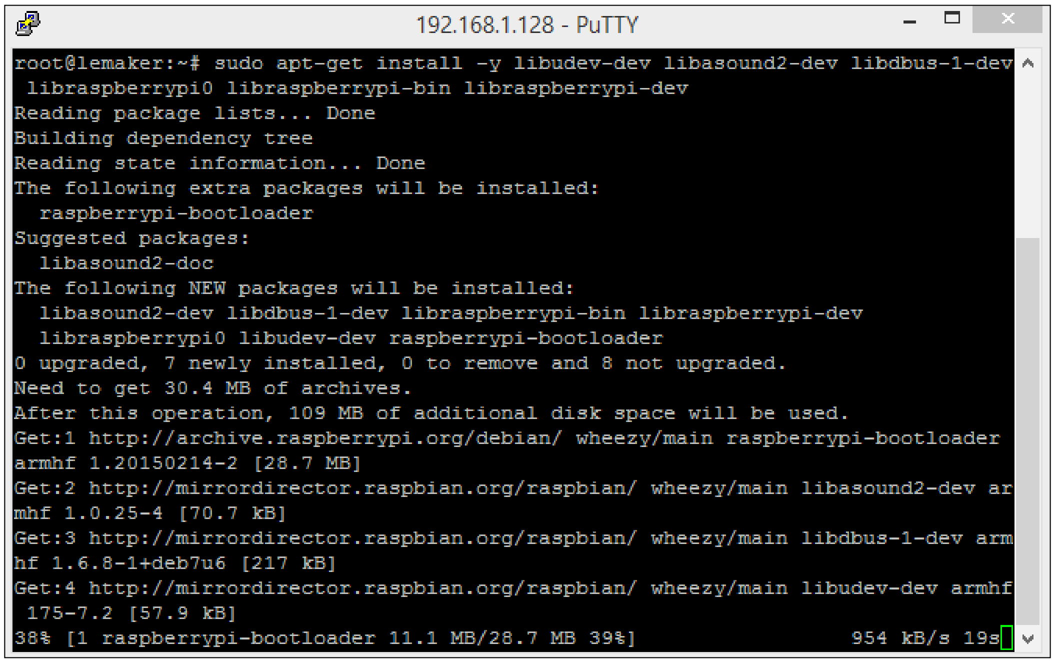
The `libudev-dev` package is particularly important because if you skip it, you may not have keyboard support inside the software later. We can now compile and install SDL2. First we will download SDL2 and unpack it, as follows:

```
wget http://libsdl.org/release/SDL2-2.0.1.tar.gz
tar xvfz SDL2-2.0.1.tar.gz
rm SDL2-2.0.1.tar.gz
pushd SDL2-2.0.1
```

Now we will compile the library, as follows:

```
./configure --disable-video-opengl --host=arm-raspberry-linux-gnueabihf
make
sudo make install
popd
```

We will get some output that looks like this:



```
root@lemaker:~# sudo apt-get install -y libudev-dev libasound2-dev libdbus-1-dev
 libraspberrypi0 libraspberrypi-bin libraspberrypi-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  raspberrypi-bootloader
Suggested packages:
  libasound2-doc
The following NEW packages will be installed:
  libasound2-dev libdbus-1-dev libraspberrypi-bin libraspberrypi-dev
  libraspberrypi0 libudev-dev raspberrypi-bootloader
0 upgraded, 7 newly installed, 0 to remove and 8 not upgraded.
Need to get 30.4 MB of archives.
After this operation, 109 MB of additional disk space will be used.
Get:1 http://archive.raspberrypi.org/debian/ wheezy/main raspberrypi-bootloader
armhf 1.20150214-2 [28.7 MB]
Get:2 http://mirrordirector.raspbian.org/raspbian/ wheezy/main libasound2-dev ar
mhf 1.0.25-4 [70.7 kB]
Get:3 http://mirrordirector.raspbian.org/raspbian/ wheezy/main libdbus-1-dev arm
hf 1.6.8-1+deb7u6 [217 kB]
Get:4 http://mirrordirector.raspbian.org/raspbian/ wheezy/main libudev-dev armhf
 175-7.2 [57.9 kB]
38% [1 raspberrypi-bootloader 11.1 MB/28.7 MB 39%]                954 kB/s 19s
```

We might as well remove the folder since the library is installed. This can be achieved by using the following command:

```
rm -rf SDL2-2.0.1
```

Now we can move on to actually compile EmulationStation. We will pull the Git repository and check out the unstable branch first, as follows:

```
git clone https://github.com/Aloshi/EmulationStation
cd EmulationStation
git checkout unstable
```

If you are following the instructions on the EmulationStation website, you will notice that they pass in different arguments to the `cmake` command. This will produce errors. In order to properly build EmulationStation, we will need to use the following commands:

```
cmake . -DFREETYPE_INCLUDE_DIRS=/usr/include/freetype/ -DCMAKE_CXX_
COMPILER=g++-4.7
```

Now we can compile the software, as follows:

```
make
sudo make install
```

You can launch EmulationStation by using the command line and typing `emulationstation`. Now we will have to reverse the changes that we made to the `config.txt` file under `boot/`, as follows:

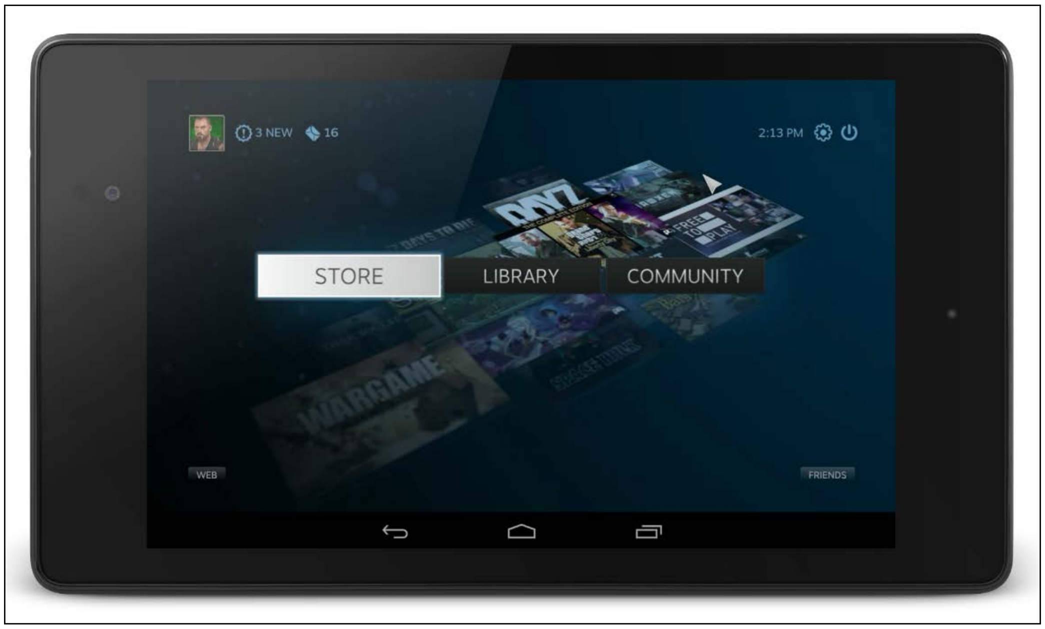
```
sudo nano /boot/config.txt
# Remove the line we added earlier.
```

You now have a fully functional system.

Streaming from your PC

PC gaming is considered one of the best experiences. One might have wished for a Steam OS port on the Banana Pi, but unfortunately, at the time of writing this book, there is currently no such port (somebody should fix this). There are other ways to stream your games from your gaming rig to your Banana Pi, though. Limelight streaming is an open source project that was developed to stream your streaming-enabled games just like Steam OS.

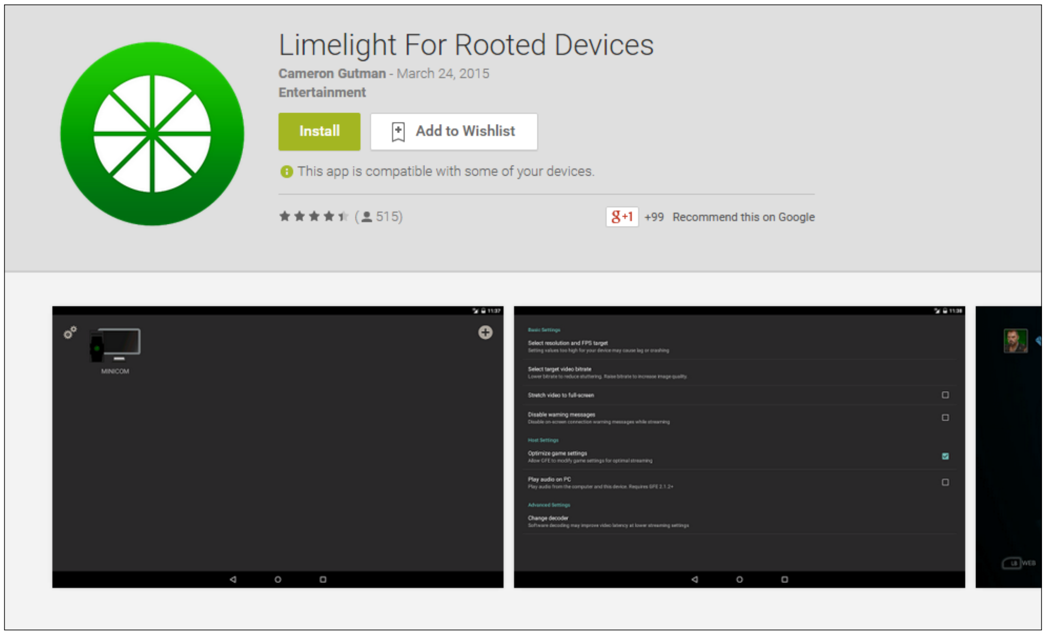
The following image shows Limelight:



As you can see from the preceding image, the Steam UI is being streamed to an Android tablet. Essentially, the Banana Pi running Android is also an Android tablet. So, we will make that work.

Limelight is nice because it has controller support and works on multiple platforms. Also, it has an extremely low latency, is open source, and free. It is only fitting for an open-hardware board that we stick to open-hardware solutions.

There is a Limelight Pi Raspberry Pi image available, but it is not ported to any version of the Banana Pi at the time of writing this. That being said, we can install the Android app, which is available:


Limelight For Rooted Devices

Cameron Gutman - March 24, 2015

Entertainment

Install | Add to Wishlist

This app is compatible with some of your devices.

★★★★½ (515) | g+1 +99 Recommend this on Google

Basic Settings

Select resolution and FPS target
Setting values too high for your device may cause lag or crashing

Select target video bitrate
Lower bitrate to reduce stuttering. Raise bitrate to increase image quality.

Stretch video to full-screen ☐

Disable warning messages ☐
Disable on-screen connection warning messages while streaming

Host Settings

Optimize game settings ☑
Allow GFE to modify game settings for optimal streaming

Play audio on PC ☐
Play audio from the computer and this device. Requires GFE 2.1.2+

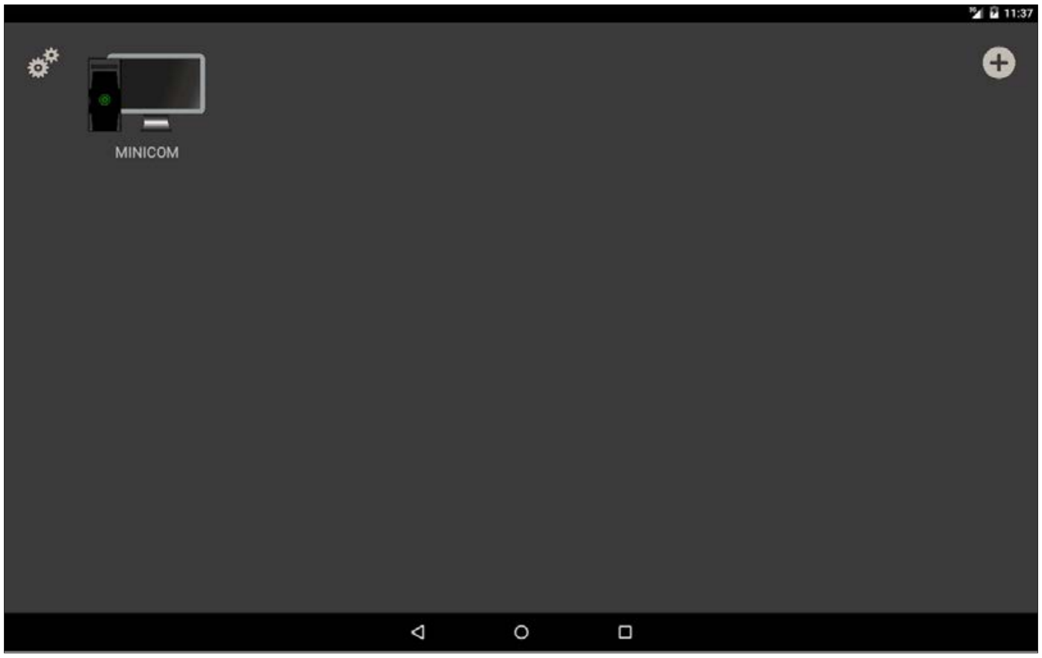
Advanced Settings

Change decoder
Software decoding may improve video latency at lower streaming settings

You can get the app from the Google Play Store by searching `Limelight for Rooted Devices`. The Android image that is available for the Banana Pi is rooted by default, so no worries there.

The app setup is fairly simple. There are only a couple of interfaces. The first interface is the main screen, which shows a list of the computers that are currently saved. You can either search for the computers that are currently streaming or add another computer by entering its IP address manually. You can add new computers by clicking on the **+** button in the top corner.

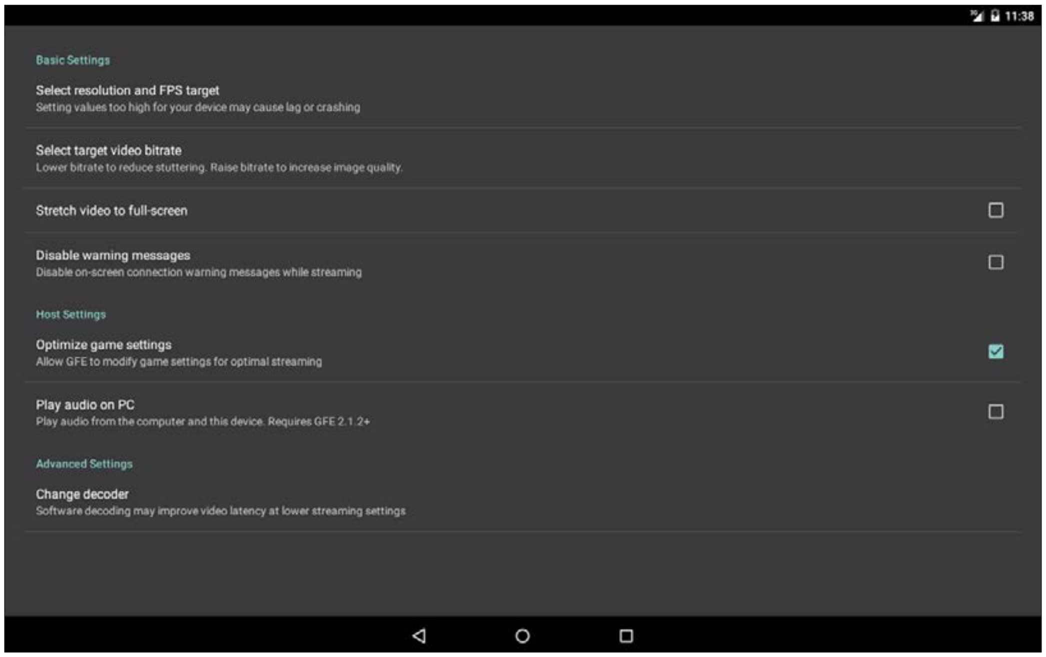
The following screenshot shows the main interface of the Limelight app:



You can have multiple computers running stream games, but you can only stream one game at a time to the devices. If you go by this route, you should ensure that the computer running the game won't be used for anything else during this time.

There is another important interface on the Limelight app. The setting page can be accessed through the gears at the top left corner of the main UI. Here, you will find settings for a basic device configuration such as video bitrate, stretching a video to fit the entire screen, and disabling warning messages.

You can also enable multiple controller support and optimize the host settings, for example the game settings, if you want the audio to stream or play from the host computer. The following screenshot shows the settings page for the app:



11:38

Basic Settings

Select resolution and FPS target
Setting values too high for your device may cause lag or crashing

Select target video bitrate
Lower bitrate to reduce stuttering. Raise bitrate to increase image quality.

Stretch video to full-screen ☐

Disable warning messages ☐
Disable on-screen connection warning messages while streaming

Host Settings

Optimize game settings ☑
Allow GFE to modify game settings for optimal streaming

Play audio on PC ☐
Play audio from the computer and this device. Requires GFE 2.1.2+

Advanced Settings

Change decoder
Software decoding may improve video latency at lower streaming settings

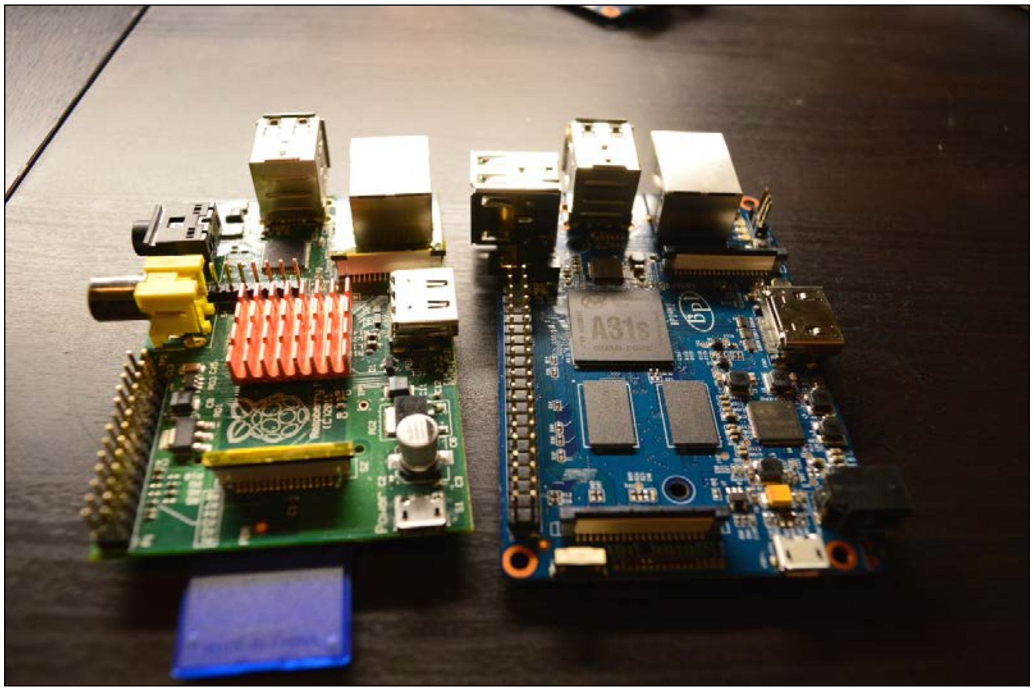
The additional settings include the UI settings that let you control not only whether you want to see the items as a list, but also whether you want to use small icons. The rest of the app is simple. Start playing a game or stream the game on the gaming computer and connect it to the Banana Pi.

Summary

Gaming is important to so many people that it only makes sense that there are a lot of options to turn your single-board computer into a streaming or an emulator PC. The projects that we explored in this chapter are all open source, which is awesome. You can see that the community has really come together to build an awesome experience by using one of these devices.

The ability to run Android on the Banana Pi is awesome for many reasons. You have an access to a lot of apps that you normally wouldn't have because of the ARM architecture. Also, you can do stuff such as downloading top-quality games or even streaming your PC games directly to your TV.

By now, we have covered many topics in this book, from taking the Banana Pi out of the box and building cool projects with it, to relaxing with your Banana Pi. You can see how similar all of these boards are physically, but when you look under the hood, there is a lot going on.



The Banana Pi is one of the most versatile Linux computers on the market today. It is a good solution for hobbyists and can be used in large-scale projects.

In the future, on ARM-based single-board computers and the Banana Pi computer, I am currently looking to develop solutions for small businesses that utilize these computers. I even work with the local college, where I develop modular classrooms that use the Banana Pi as an educational tool that brings education to the remote communities.

Index