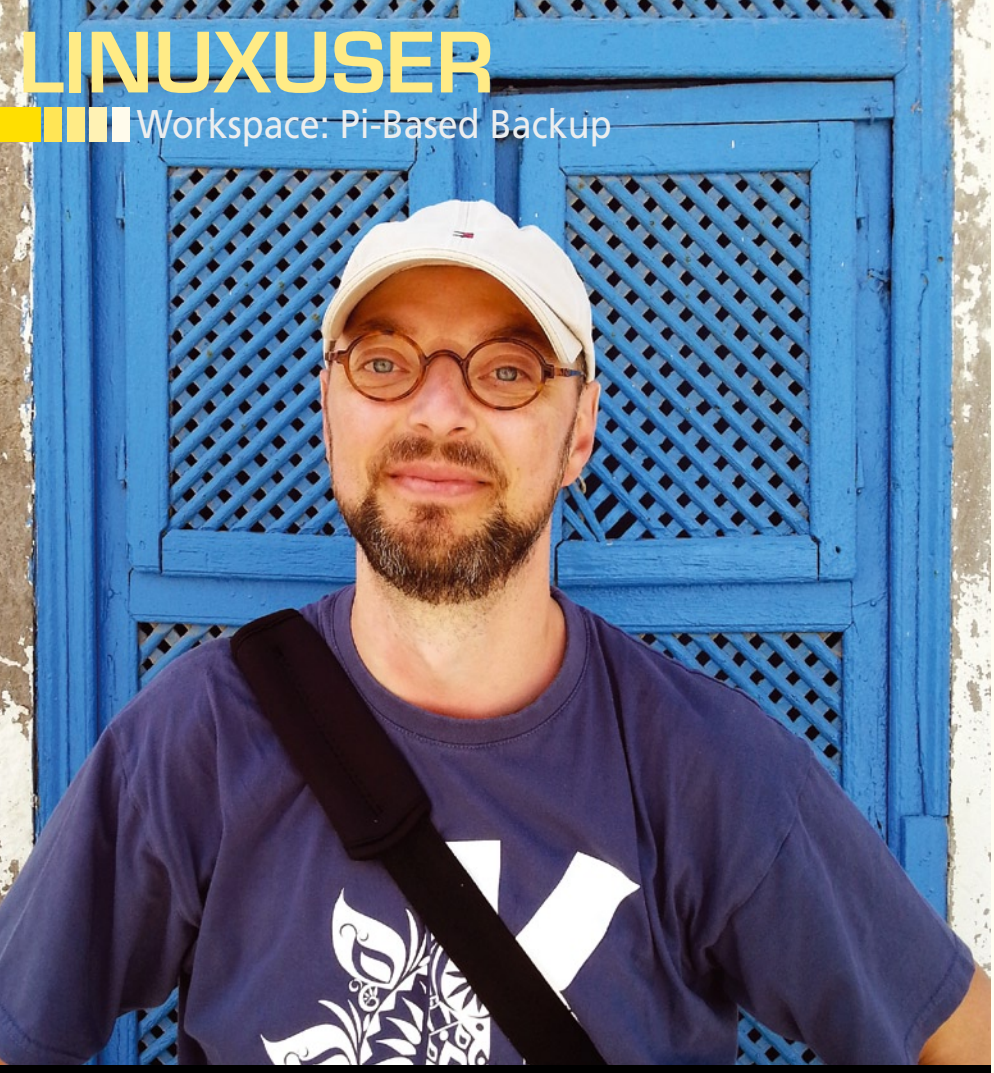

Build a Raspberry Pi-based backup device

Tiny Backup Box

With some creativity and a little scripting, you can easily turn your Raspberry Pi into an effective backup device.

By Dmitri Popov

DMITRI POPOV

Dmitri Popov has been writing exclusively about Linux and open source software for many years, and his articles have appeared in Danish, British, US, German, Spanish, and Russian magazines and websites. Dmitri is an amateur photographer, and he writes about open source photography tools on his Scribbles and Snaps blog at *scribblesandsnaps.com*.

Backup is crucial, especially when you are traveling. This is particularly true for photos: To keep your snaps safe, you ought to have at least one backup set of your precious photos. That's why a light, flexible, and inexpensive backup tool can be an indispensable tool in your travel bag. Several backup solutions are available on the market, but with a Raspberry Pi, you can build your own backup device and learn a few useful tricks and skills in the process.

Required Components

To begin, you will need a Raspberry Pi along with the Raspbian Linux distro installed on an SD card. Although you can use a standard micro-USB charger to power Raspberry Pi, you might want to invest in an external battery pack to make the solution more portable. Finally, you need a high-capacity USB key for storing backups. In theory, you could use a USB hard disk, but, because it must be connected to Raspberry Pi through a powered USB hub, this approach would make the setup unwieldy.

Going the Software Route

The easiest way to transform Raspberry Pi into a backup device is to use the excellent gPhoto2 [1] software available in the Raspbian software repository. To install it, you use the

```
sudo apt-get install gphoto2
```

command. Although the older version from the repository should do the trick, you might want to compile the latest release of gPhoto2 from the source code because it provides support for new camera models along with a handful of fixes and improvements.

Compiling from the source code can be a daunting task, but the *gphoto2-updater* Bash shell script automates the entire process [2]. Grab the script from the project's GitHub repository by clicking on the *ZIP* button, unpack the downloaded archive, make the `gphoto2-updater.sh` script executable with the

```
chmod +x gphoto2-updater.sh
```

command, then issue the

```
./gphoto2-updater.sh
```

command to run the script.

You also need to install two additional packages on Raspberry Pi: usbmount and ExifTool. The former allows Raspberry Pi to mount and unmount USB devices automatically, whereas the latter is necessary for processing downloaded files. To install both packages, enter:

```
sudo apt-get install usbmount ➥
    libimage-exiftool-perl
```

With all the pieces in place, you are ready to proceed. The first order of business is to write a Bash shell script that transfers photos from the camera connected to the Raspberry Pi. Run the `nano`

get-all-files.sh command to create an empty file and open it in the nano text editor, then enter the following code:

```
mkdir "`date --iso-8601`" && cd $_
gphoto2 --get-all-files
exiftool -r -d %Y%m%d-%H%M%S.%%e "-FileName<DateTimeOriginal" .
```

The script is not particularly complicated. First it creates a directory using the current date in the ISO format as its name and then switches to the created folder. The script then pulls photos from the camera with the gphoto2 --get-all-files command and uses ExifTool to rename them on the basis of date and time info from Exif metadata.

Save the script and make it executable with chmod +x get-all-files.sh. Next, connect the camera to Raspberry Pi, turn the camera on, and run the script with the command ./get-all-files.sh to transfer photos.

Although this simple script does the job, it's far from ready. To run the script, you must be able to control Raspberry Pi either directly or via an SSH connection. Ideally, it should kick in automatically as soon as it detects the connected camera. To make this happen, you need to add code that constantly checks for the camera and transfers photos as soon as it detects it:

```
DEVICE=$(gphoto2 --auto-detect | grep usb | cut -b 36-42 | sed 's/,/\//')
while [ -z ${DEVICE} ]
        do
        sleep 1
        DEVICE=$(gphoto2 --auto-detect | grep usb | cut -b 36-42 | sed 's/,/\//')
done
```

By default, the script performs backup in the current working path (i.e., on the SD card). If you prefer to use a USB key as a backup destination, you need to tweak the script so it switches to the appropriate mountpoint. This is where the usbmount tool comes into the picture: It automatically mounts a connected USB device (in this case, the USB key) at the first available /media/usb mountpoint.

If the USB key is the only storage device connected to Raspberry Pi, usbmount mounts it at the /media/usb0 point, so you need to add the cd /media/usb0/ command to the script. Finally, you might want to add the halt command, so the script automatically shuts down Raspberry Pi after the photos have been transferred and processed. You can see the script in its entirety in Listing 1.

One more step is needed, which is to configure Raspberry Pi to run the script automatically. To do this, open the inittab file in nano using the sudo nano /etc/inittab command and locate the following line:

```
1:2345:respawn:/sbin/getty --noclear 38400 tty1
```

Add the --autologin pi option to it as follows and save the file:

```
1:2345:respawn:/sbin/getty ➥
   --autologin pi --noclear 38400 tty1
```

Next, run the nano .bash_profile command and add the sudo ./get-all-files.sh line to the opened .bash_profile file. Save the changes, and you are done. Reboot Raspberry Pi, plug in, and turn on the camera, and the script should do the rest.

If your Raspberry Pi is connected to the Internet, you can modify the script to perform off-site backup. To do this, install the Rsync tool on Raspberry Pi and the re-

LISTING 1: get-all-files.sh Shell Script

```
01 #!/bin/bash
02 DEVICE=$(gphoto2 --auto-detect | grep usb | cut -b 36-42 | sed 's/,/\//')
03 while [ -z ${DEVICE} ]
04         do
05         sleep 1
06         DEVICE=$(gphoto2 --auto-detect | grep usb | cut -b 36-42 | sed 's/,/\//')
07 done
08 cd /media/usb0/
09 mkdir "`date --iso-8601`" && cd $_
10 gphoto2 --get-all-files
11 exiftool -r -d %Y%m%d-%H%M%S.%%e "-FileName<DateTimeOriginal" .
12 halt
```

mote backup server. Next, you need to configure a passwordless SSH login for the remote server. On Raspberry Pi, run the following command to generate a key pair:

```
ssh-keygen -t dsa
```

Don't enter any password when prompted, and confirm the default choices by pressing the Enter key. Then, copy the generated public key to the remote server using the command below (replace `<user>` with the actual user name and `<remotehost>` with the IP address or domain name of the remote server):

```
ssh-copy-id -i .ssh/id_dsa.pub <user>@<remotehost>
```

Finally, add the following command to the `get-all-files.sh` script:

```
rsync -avhe ssh --delete /<path/to/source/dir> <user>@<remotehost>:/<path/to/target/dir>
```

Don't forget to replace the placeholders with actual paths.

When running on its own, Raspberry Pi provides practically no feedback, so you might have trouble telling when the script is done. Of course, Raspberry Pi shuts down automatically after the backup has been performed, but figuring out whether the device is still running can be tricky. One way to solve this problem is to tweak the script, so it plays an MP3 file before issuing the `halt` command.

To add this functionality, you can install the mpg123 player with the `sudo apt-get install mpg123` command. Next, insert the `mpg123 sound.mp3` line before the `halt` command in the `get-all-files.sh` script (replace `sound.mp3` with the actual name of the sound file). Plug earphones or a speaker into Raspberry Pi's audio jack, and you should hear the sound before the system shuts down.

Adding Some Hardware

The backup solution described above does the trick, but it can be improved even further. For example, you can add buttons to trigger the backup script and shut down Raspberry Pi, and you can install an LED for visual feedback. All of this is possible thanks to Raspberry Pi's GPIO pins, which can control a variety of inputs and outputs. For this project, you'll need a few additional components, including a breadboard, a handful of jumper wires (both male-male and female-male), two 10K resistors and a single 10-ohm resistor, two push buttons, and an LED.

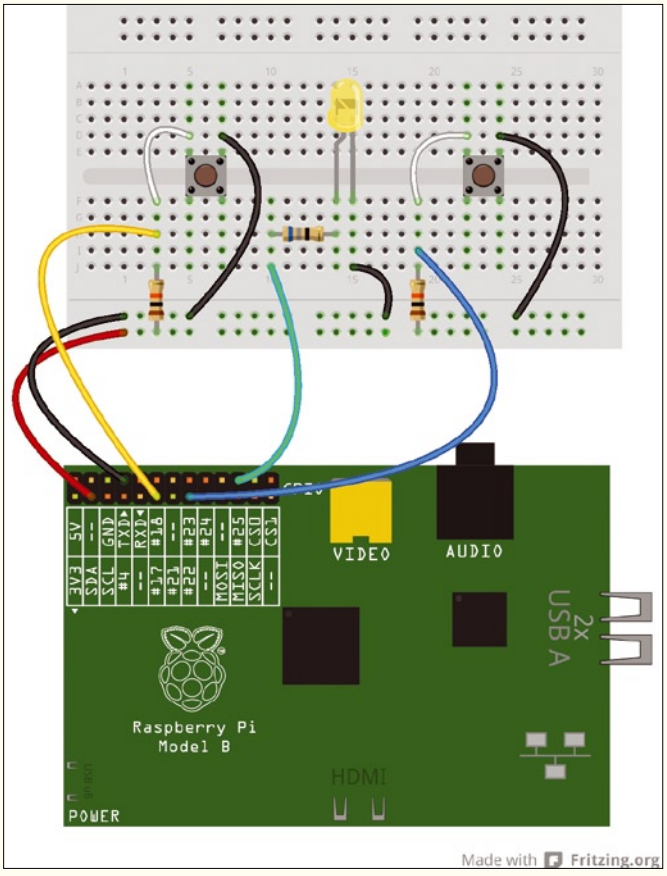

Figure 1: Wiring diagram.

Start by wiring the components as shown in Figure 1. Two separate circuits contain a pull-up 10K resistor and a push button connected to pins 17 and 23 and a circuit with a 68-ohm resistor and an LED connected to pin 25. Next, you can install the `RPi.GPIO` Python module using the `sudo apt-get install python-dev python-rpi.gpio` command. Before you start working on a Python script that reads GPIO inputs and performs the required actions, you should write a simple Bash shell script that does the actual backup:

```
#!/bin/bash
mkdir "`date --iso-8601`" && cd $_
gphoto2 --get-all-files --filename
exiftool -r -d %Y%m%d-%H%M%S.%%e "-FileName<DateTimeOriginal" .
```

Save the script under the `get-all-files.sh` name. Now you can use the code in Listing 2 to create a Python script that does several things. First, it initializes pins 17 and 23 for input and pin 25 for output; then, it enables pin 25, thus turning on the LED (this indicates that the script is running and ready to accept user input). The script then waits for input on pins 17 and 23.

When the user presses the push button wired to pin 17, this triggers the `os.system ('./get-all-files.sh')` call, which executes the Bash shell backup script and turns off the LED. When the script detects a signal on pin 25, it shuts down the system.

Save the script under the `fetch.py` name and make it executable using the

```
chmod +x fetch.py
```

command. Then, run the `nano .bash_profile` command and replace the `sudo ./get-all-files.sh` entry added previously with `sudo ./fetch.py`. Save the changes, reboot Raspberry Pi, and wait until the LED turns on. Plug in and turn on the camera, and press the first push button to perform backup (Figure 2). When the LED turns off, press the second push button to shut down the system.

LISTING 2: fetch.py Python Script

```
01 #!/usr/bin/env python
02 from time import sleep
03 import os
04 import RPi.GPIO as GPIO
05 GPIO.setmode(GPIO.BCM)
06 GPIO.setwarnings(False)
07 GPIO.setup(17, GPIO.IN)
08 GPIO.setup(23, GPIO.IN)
09 GPIO.setup(25, GPIO.OUT)
10 GPIO.output(25, True)
11 while True:
12         if (GPIO.input(17) == False ):
13                 os.system('./get-all-files.sh')
14                 GPIO.output(25, False)
15         if (GPIO.input(23) == False):
16                 os.system('sudo halt')
17         sleep(0.1);
```

Final Word

I've described two ways to transform a Raspberry Pi into a backup device. With some creative thinking and hacking, you can easily tweak and improve the described recipes, for example, by writing a script that processes the transferred photos and publishes them on the web. Also, you can configure Raspberry Pi to send email notification when the backup operation has completed. All code and files mentioned in this article ares available on GitHub [3].

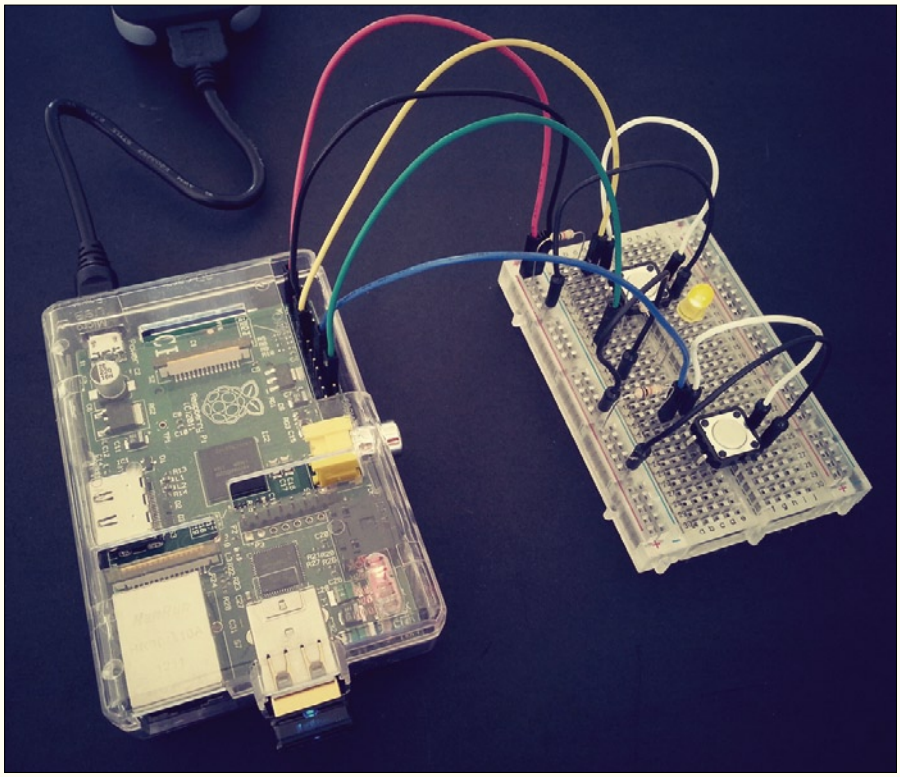

Figure 2: Raspberry Pi photo backup prototype in action.

INFO

[1] gPhoto2: *gphoto.sourceforge.net*
[2] gphoto2-updater Bash shell script: *github.com/dmpop/gphoto2-updater*
[3] Code and files from the article: *github.com/dmpop/rpi-photo*