BrewPi Documentation

Release 0.3.0

Elco Jacobs

Oct 31, 2017

Contents

This documentation is written in reStructured text format and part of the brewpi-www repository. If you see something that could use some updating, please let me know! Even better: fix it yourself and send me a pull request on GitHub!

The steps below will help you to go from a blank SD card and an unprogrammed Arduino to a working BrewPi setup. Install and update scripts are available to do most of the work for you. After flashing Raspbian Wheezy onto your Pi, these scripts will automate 80% of the process. If you really want to, you can still do your setup manually or just check the manual setup guide to see what the install script will do for you.

CHAPTER 1

Basic configuration of your Raspberry Pi

This step by step guide will help you to set everything up on your Raspberry Pi and Arduino.

Installing Linux on your Raspberry Pi

The easiest cross-platform way to install Linux on your Raspberry Pi is using the NOOBS installer.

1. Go to the Raspberry Pi download page
2. Download the NOOBS zip file.
3. Download SDFormatter (link on download page too).
4. Format your SD card with SDFormatter. The defaults are okay.
5. Extract the contents of the NOOBS zip file to your SD card.
6. Put the SD card in your Raspberry Pi and boot it.
7. You will get a menu to choose which OS you want to install, choose Raspbian.

Windows users can also use Win32DiskImager to write the image to the SD card. If you have an old image on your card and cannot remove the partition with Windows, SD Formatter can help you to format the SD card before writing the image.

A list of recommended SD cards can be found at http://elinux.org/RPi_SD_cards The default login credentials for the image are username `pi` and password `raspberry`. You can change the password in the next step.

Configuring your system

SSH is enabled by default in the image you installed. If you have an Ethernet cable connected, you may SSH into the Pi to begin your configuration. The wired ethernet port is set to receive a dynamic IP address using DHCP. To find out which IP it got, use your router's web interface.

To connect via ssh, you will need an SSH client such as Putty Connect to the IP address your Pi is using (port 22, SSH). When you save your session, you can also save your password.

Alternatively, you may also plug in a keyboard and monitor and use the on screen menu to configure the basic settings.

If the basic system settings menu doesn't come up automatically, type:

```
sudo raspi-config
```

Setting up WiFi

I am using an Edimax EW-7811un WiFi dongle. The latest image has built in support for this dongle. I can highly recommend this dongle. The dongle is based on a Realtek 8192CU or 8188CUS, but the 8192cu driver that ships with Raspbian works for both.

To improve the stability of your WiFi, it is recommended to disable power management:

1. Create a new file `8192cu.conf` in `/etc/modprobe.d/`:

```
sudo nano /etc/modprobe.d/8192cu.conf
```

1. Add this line to the file and save:

```
options 8192cu rtw_power_mgnt=0 rtw_enusbss=0
```

Alternatively, you can add a line to your crontab to ping your router every minute to keep the connection alive. Open your current users crontab with:

```
crontab -e
```

And add this line (replace the IP with your routers IP address, the standard gateway address in your IP settings):

```
* * * * * ping -c 1 192.168.0.1
```

Setting up a static IP address

You probably want your Pi to always have the same IP address, so after setting up your WiFi change your interfaces file to a static IP address. To edit your interfaces, you can run:

```
sudo nano /etc/network/interfaces
```

My /etc/network/interfaces file looks like this:

```
auto lo
iface lo inet loopback
iface eth0 inet dhcp
allow-hotplug wlan0
auto wlan0
iface wlan0 inet static
address 192.168.0.6
netmask 255.255.255.0
gateway 192.168.0.1
wpa-ssid "YOUR_SSID"
wpa-psk "YOUR_PASSPHRASE"
```

The right IP addresses depend on your home network setup. You can run ifconfig before editing the interfaces and write the automatically assigned addresses down. However, it is recommended to pick a static Ip address that is outside of your router's DHCP range.

Finally, you will need to restart your network interfaces for these changes to occur:

```
sudo service networking restart
```

Updating programs

Keep your programs up to date with these commands:

```
sudo apt-get update
sudo apt-get upgrade
```

Updating firmware

Make sure you also have the latest firmware version, and stay up to date using rpi-update by Hexxeh. Firmware updates will often fix instability issues, so make sure you run one.

CHAPTER 2

Automated installation of BrewPi

We created scripts to install and update BrewPi, these scripts are part of the brewpi-tools repository on GitHub. The install script will install all dependencies, set up users and permissions, download the latest code base and setup a CRON job. Just answer the questions on the screen.

Getting and running the install script

You can get the install and update scripts by cloning the BrewPi tools repository from GitHub. The scripts use git to check for self-updates, so keep them inside the repository. You can put the brewpi-tools dir anywhere you like, the instructions here assume the pi user's home directory.

Use the following commands to clone the repository to the your current user's home dir and to run the install script:

```
git clone https://github.com/BrewPi/brewpi-tools.git ~/brewpi-tools
sudo ~/brewpi-tools/install.sh
```

Now just follow the instructions on the screen. The install script will remove everything in the install directories you pick (after making a backup). So if you want a fresh start, you can always rerun the installer.

After the install has finished

If the installation was successful, your next step is to program the Arduino via the web interface. You can find the instructions *here*.

After that, your last step will be to *to set up your devices in the device manager*.

CHAPTER 3

Manual installation of BrewPi

This section describes how to manually setup BrewPi on your Raspberry Pi: how to install dependencies, the web server, Python, users and permissions, cloning the files from GitHub, modifying the config files and setting up a CRON job.

Setting up your LAMP web server for BrewPi

LAMP stands for Linux, Apache, MySQL, PHP: the applications that make up your web server. This article explains how to install all of them on your pi.

Install Apache2

To install Apache2, execute the following commands:

```
sudo apt-get update
sudo apt-get install apache2
sudo apt-get install libapache2-mod-php5 php5-cli php5-common php5-cgi
```

To test your server, you can now visit http://your-rpi-ip/. By default, the root of your web server is `/var/www/`

Install PHP

To install PHP, execute the following commands:

```
sudo apt-get install php5
```

Once you have installed PHP, you can test it by creating a new php file in your `/var/www/` directory. Open the text editor nano to create the file:

```
sudo nano /var/www/phpinfo.php
```

This is what you want to put inside your file:

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

To exit nano and save your file, you can use `ctr-x`, to write the file but keep nano open, you can use `ctrl-o`.

For security reasons, your web server runs as a separate user under Linux, the `www-data` user. Your files in your /var/www/ directory should be owned by this user, otherwise your web server will not have the permissions to access the files. You can change owner of everything in your `/var/www/` directory to the `www-data` group and user with the following command. The `-R` flag recursively chowns all files and subdirectories.

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

You can test your PHP installation by viewing the result of your new file in a browser: http://your-rpi-ip/phpinfo.php. If that worked, please delete the file:

```
sudo rm /var/www/phpinfo.php
```

If you ever need to restart your web server, you can use this command:

```
sudo /etc/init.d/apache2 restart
```

Install MySQL

The current version of BrewPi doesn't use MySQL. It is recommended to not install it. If you want MySQL anyway, you can install it with these commands:

```
sudo apt-get install mysql-server mysql-client php5-mysql
```

To manage your databases from a web interface, you can install PHPMyAdmin:

```
sudo apt-get install libapache2-mod-auth-mysql php5-mysql phpmyadmin
```

Remember to select Apache2 with the space bar when it comes up! After installation, reboot and test: http://your-rpi-ip/phpmyadmin. Reboot with the command:

```
sudo shutdown -r now
```

To easily see when your pi is back online, create a shortcut in your quick launch bar with the following target (for the correct IP address of course):

```
C:\Windows\System32\cmd.exe /c "ping -t 192.168.0.6"
```

That will open a command window that keeps pinging your Raspberry Pi.

If you mess up the installation of MySQL, because you didn't listen and did not expand your root partition, it can be a bitch to remove and re-install. The following worked for me eventually:

```
sudo apt-get autoremove --purge mysql-server mysql-server-5.5 mysql-common
sudo rm /var/lib/mysql/ -rf
sudo rm /etc/mysql -rf
sudo apt-get install -f mysql-server
```

Installing python requirements

The Raspbian Wheezy images comes with python 2.7 already installed, but BrewPi uses a few python modules that you have to install yourself:

- pySerial For communication with the Arduino.
- simplejson To encode and decode JSON objects. Easier to use than the default JSON module.
- ConfigObj For configuration files support.
- psutil To view running processes and close/kill them from within python
- python-git A Python library to work with git repositories. Used by the update script.

All 5 modules can be easily installed with apt-get:

```
sudo apt-get install python-serial python-simplejson python-configobj python-psutil␣
↪python-git
```

Finally, the python script can reprogram the Arduino, but it needs avrdude to do so. Install arduino-core, which includes avrdude.

```
sudo apt-get install arduino-core
```

That's it, everything is ready for running the BrewPi script. Let's get it from GitHub in the next step.

Setting up users and permissions

For security reasons, we will run the BrewPi python script under its own user account: an account without sudo rights. We will set up the `brewpi` user and group now. The web interface will run under the user and group `www-data`. The www-data user already exists. We will add the `brewpi` user with `useradd`, which automatically creates the group `brewpi` as well. Additionally, we are adding the user `brewpi` to the `www-data` and `dialout` groups (needed for access to the /var/www/ dir and serial ports, respectively). Then, we set the password for the brewpi user with passwd.

```
sudo useradd -m -k /dev/null -G www-data,dialout brewpi
sudo passwd brewpi
```

Now, verify your work:

```
id brewpi
```

You should see something similar to:

```
uid=1001(brewpi) gid=1002(brewpi) groups=1002(brewpi),20(dialout),33(www-data)
```

The python script will reside in the brewpi home directory. It will log data to the ./data subdirectory, keep settings in ./settings and it will copy everything the web interface needs to know to /var/www/ and chown it to www-data. By doing it this way, the www-data user does not have to have any rights outside its own directory. To allow the brewpi user to write to the directories owned by www-data, we will have to add it to the www-data group. We will also add the pi user to both groups to make it easier to work with the files.

```
sudo usermod -a -G www-data pi
sudo usermod -a -G brewpi pi
```

To make sure that all newly created files in the www-data directory have www-data as group, even when they are created by the brewpi user, we set the sticky bit on the www-data directory and all its sub directories. We'll set the sticky bit for the brewpi home directory as well. Run the following commands:

```
sudo chown -R www-data:www-data /var/www
sudo chown -R brewpi:brewpi /home/brewpi
sudo find /home/brewpi -type f -exec chmod g+rwx {} \;
sudo find /home/brewpi -type d -exec chmod g+rwxs {} \;
sudo find /var/www -type d -exec chmod g+rwxs {} \;
sudo find /var/www -type f -exec chmod g+rwx {} \;
```

These commands do the following things:

- Set the ownership of all files and subdirectories to brewpi and www-data (first two lines)
- Give the group all permissions on all files (third and fourth line)
- Give the group all permissions and set the sticky bit on all directories (fifth and sixth line).

Fixing permissions issues

If you run into permission issues later, you can use a script included with the brewpi-script repository to fix it. This could happen for example when you did not run git as the brewpi user or the www-data user, which results in the owner of the files being `pi` or `root`. This will cause errors when the web interface or script tries to access files. This script just executes the commands above. Run it with:

```
sudo /home/brewpi/utils/fixPermissions.sh
```

Starting and stopping the python script

There is a button in the web interface to start and stop the brewpi script. But allowing the www-data user to start python scripts would create a huge security risk. This is solved by running a CRON job: every minute the system checks whether the script should be running and starts it when it does. This way the www-data user only has to create/remove a file in the web directory. We will set this up after getting the BrewPi files from Git.

Using Git for BrewPi

BrewPi uses Git, a distributed version control system, and hosts its source code on GitHub. GitHub makes it easy to share code and to collaborate with other developers. To download BrewPi to your Raspberry Pi, we will clone the BrewPi repositories from the server to your device. A git clone copies the current version of all files, but also all other versions and the complete history in the .git directory.

To use Git, you will first have to install it with:

```
sudo apt-get install git-core
```

BrewPi is divided into 3 projects, which are all hosted on GitHub. To make you understand a bit better how BrewPi is build up, here is a short explanation of what each part does.

The Arduino code: brewpi-avr

The Arduino runs all temperature control, reads sensors, switches actuators and drives the display. BrewPi can run standalone on the Arduino and maintain a constant beer temperature, so if your Raspberry Pi crashes, your beer is still safe.

If you just use the standard BrewPi shields and do not want to mess with the code, you can just download the latest HEX file to upload to your Arduino from the BrewPi servers at http://dl.brewpi.com.

The brewpi-avr repository can be found at https://github.com/BrewPi/brewpi-avr. This repository contains the code that runs on the Arduino. Because the Arduino IDE is basically just a crappy notepad with an upload button, we are using Atmel Studio to write our code. We just include the Arduino libraries in the project. Atmel Studio is built with the Visual Studio 2010 shell, which makes coding a lot faster and less frustrating.

The python script: brewpi-script

This repository contains the BrewPi Python script. This python script connects to the Arduino and listens to incoming connections from the web interface. These are the main functions the script provides:

- Periodically request data from the Arduino and write it to files for displaying it in graphs
- If BrewPi is running in profile mode, the python script reads a profile and continuously updates the beer temperature setting
- The python script can reprogram the Arduino
- The python script manages settings and forms an API layer between the Arduino and the web interface.

The brewpi-script repository can be found at https://github.com/BrewPi/brewpi-script.

Cloning the remote repository

The following command will clone the brewpi-script repository into `/home/brewpi`, an empty directory you should have created in the previous step. The python script will run as the brewpi user, so it is best if we run Git as the brewpi user too, so all new files are owned by the brewpi user. We do this by adding `sudo -u brewpi` to the command.

```
sudo -u brewpi git clone https://github.com/BrewPi/brewpi-script /home/brewpi
```

After the git clone, Git will remember where you got the files from: the URL above is stored as the default remote, named `origin`. There are different branches within the repository, the most import ones being `master` and `develop`. `master` will contain the latest stable code and is checked out by default. `develop` will have the latest updates, but it is not guaranteed to be stable. So for normal uses, it is recommended to just stick to `master`.

Switching between branches

To switch to a different branch, you use `git checkout`:

```
cd /home/brewpi
sudo -u brewpi git checkout develop
```

Updating to the latest version

With the files in version control, updating is easy too:

```
cd /home/brewpi
sudo -u brewpi git pull
```

Git pull is a combination of two commands: `git fetch`, which will update your local copy of the remote, and `git merge`, which merges the changes from the remote with your local files.

Dealing with local changes

If you have changed files locally, the commands above will likely fail because of a conflict. `git stash` comes in handy in that case. `git stash` stashes your local changes away, so you have an unmodified copy of the latest commit again.

```
sudo -u brewpi git stash
```

After the stash you can do the pull or checkout you wanted to do. If you want your local changes back afterwards, you can get them back from the stash again with `git stash pop`.

```
sudo -u brewpi git stash pop
```

Resetting to the latest commit

If you have messed up your local copy and want to reset everything to the latest commit:

```
sudo -u brewpi git reset
```

The web interface: brewpi-www

This project is for the web interface of brewpi: PHP, JavaScript, CSS and all other files that make up the web interface. The web interface should be your primary way to interact with BrewPi: watch the graphs, change the settings and check the log files. Everything the brewpi-script does is executed from the web interface: the script has a listening socket which be web interface can send message to. The script will start actions or return data based on the message content.

The brewpi-www repository can be found at https://github.com/BrewPi/brewpi-www

Cloning the remote repository

As before, we will clone the remote repository to a local directory. In this case this is the web server's root directory: `/var/www`. The web server runs as the www-data user, so we will also run git as the www-data user this time.

The directory should be empty, so check if you have left any files from previous steps and remove them.

```
ls /var/www
sudo rm /var/www/*
sudo -u www-data git clone https://github.com/BrewPi/brewpi-www /var/www
```

The other git commands for easy copy pasting in /var/www

Here are the other git commands again, but now ran as the www-data user, so you can easily copy/paste them. They should all be run from the /var/www directory. You don't have to run these now.

```
sudo -u www-data git checkout develop
sudo -u www-data git pull
sudo -u www-data git stash
sudo -u www-data git stash pop
sudo -u www-data git reset
```

Now that we have checked out all the BrewPi files, we just have to set a few settings.

Fixing permissions after git commands

Not all permissions can be stored in git, which is why the permissions are usually wrong after working with git. You can run the fixPermissions script in the brewpi-script/utils directory for fix all permissions at once.

Modifying BrewPi config files

The defaults should work in most cases, if you run BrewPi on a Raspberry Pi and are happy with the default setup. The config files for BrewPi are split in two: default configuration and user configuration. The user config files overrule the default configuration. They are created on first use, or can be added manually.

The default config is version controlled with git, the user config files are ignored.

Editing the script config file

In the script's settings directory, the user config file is called config.cfg. By default, all settings are commented in this file. BrewPi first loads defaults.cfg, and overwrites the default settings with the user settings in *config.cfg*. Generally, the defaults will work. If you change settings in the web interface, for example the beer name or interval, these settings will be written to `config.cfg`.

If your Arduino is using a different serial port than /dev/ttyACM0, you will have add the setting to config.cfg. The altport setting is used when the normal port cannot be found.

```
port = /dev/ttyACM0
altport = /dev/ttyACM1
```

If you are not running BrewPi on a Raspberry Pi, but for instance on Windows, you will want to change a few of these parameters. For my setup on Windows I use WAMP as a web server and run the python scripts in IntelliJ IDEA (ask me for a license key).

My config.cfg file for Windows has these settings:

```
scriptPath = c:\Users\Elco\Dropbox\BrewPi\git\brewpi-script\
wwwPath = c:\wamp\www\brewpi\
port = COM7
arduinoHome = c:\arduino-1.0.4\
avrdudeHome = C:\arduino-1.0.4\hardware\tools\avr\bin\
avrsizeHome = C:\arduino-1.0.4\hardware\tools\avr\bin\
avrConf = C:\arduino-1.0.4\hardware\tools\avr\etc\avrdude.conf
```

Editing the web interface config file

In web interface, the settings in config.cfg are overruled by config_user.php, if it exists. The only setting for the web interface is the location of the scripts directory. An example file for config_user.php is included in the repository. Copy

this example file to config_user.php and make edits to the copy.

Setting up a CRON job to start the script

Since version 0.2, managing running brewpi processes integrated into the script: when BrewPi is started with the –dontrunfile option, it will check whether the file 'do_not_run_brewpi' exists in the web directory. If it exists, the script will exit. With this option, CRON can just start the script every minute. When a script is already running, the script exits immediately. The script will also check for conflicting instances of BrewPi.py at startup.

To have a cron job check BrewPi every minute, we will add a script to /etc/cron.d This script is included with the BrewPi script repository.

If you still have a line for BrewPi in the brewpi user's crontab (as used in 0.2), you will have to remove it.

The commands in this document can be automatically performed using the updateCron.sh script in your script/utils direcotry:

```
sudo /home/brewpi/utils/updateCron.sh
```

To set up the cron job manually, create the file /etc/cron.d/brewpi and add the following line:

```
* * * * * brewpi python /home/brewpi/brewpi.py --checkstartuponly --dontrunfile; [ $?␣
↪!= 0 ] && python -u /home/brewpi/brewpi.py 1>/home/brewpi/logs/stdout.txt 2>>/home/
↪brewpi/logs/stderr.txt &
```

What this line says is the following:

Piece of command	What it does
`* * * * *`	Every minute of every hour of every day of every month of every year
`python /home/brewpi/brewpi.py --checkstartuponly --dontrunfile`	Start brewpi.py to check the dontrunfile (and conflicting processes). Only do the startup checks and return 1 (do start) or 0 (do not start). This lets us do output redirection on the second call of brewpi.py in this job
`[ $? != 0 ] && python -u /home/brewpi/brewpi.py`	If the previous command didn't return 0, start brewpi.py with unbuffered output. This ensures the log files are written while the script is running.
`1>/home/brewpi/logs/stdout.txt`	Write > stdout of the process `1` to `/home/brewpi/stdout.txt`
`2>>/home/brewpi/logs/stderr.txt &`	Append >> stderr of the process `2` to `/home/brewpi/stderr.txt` and fork the command and run it in background `&`. Without `&` at the end, CRON would wait for an answer from the script.

Restart CRON after adding/editing the file:

```
sudo /etc/init.d/cron restart
```

CHAPTER 4

Programming your Arduino with BrewPi

Uploading a new hex file to your Arduino can be done straight from the BrewPi web interface. The serial port is read from the config file (`/home/brewpi/settings/config.cfg`), so make sure the settings are correct. If this file does not contain port settings, BrewPi uses the defaults from (`/home/brewpi/settings/defaults.cfg`) BrewPi defaults to ttyACM0 and when it cannot find it, it tries ttyACM1. If your serial port is one of these, you don't have to do anything.

The default should normally work (`/dev/ttyACM0`), but if your script fails to start because the serial port is not found, run this to see the candidates:

```
ls /dev/ttyA*
```

It is definitely not `ttyAMA0`, that is the internal serial port. Official Arduinos normally appear as ttyACM0 or if that is already taken, ttyACM1. Arduino clones sometimes appear as ttyUSB0 or similar. If that is the case, add these lines to (`/home/brewpi/settings/config.cfg`):

```
port = /dev/ttyUSB0
altport = /dev/ttyUSB1
```

Uploading a new HEX file to your Arduino

To program your Arduino from the web interface, take the following steps:

1. Log on to your BrewPi web interface by entering the Raspberry Pi's IP address into a web browser. The IP is displayed when the installer finishes, or by typing `ifconfig` at the shell (look for inet addr). If the page says 'script not running' you should start the BrewPi script first. This is automatically done by the CRON job within one minute if you click the 'start script' button. If you have not set up CRON jet, you can manually start the script with:

   ```
   sudo -u brewpi python /home/brewpi/brewpi.py
   ```

2. Open the maintenance panel and go to the *Reprogram Arduino* tab.

3. Download the HEX file appropriate for your setup from http://dl.brewpi.com/brewpi-avr/stable. You can also compile your own hex file with Atmel Studio. Make sure you have the right file for your Arduino type (UNO or Leonardo) and the right shield (Rev A or Rev C). Only the first 100 BrewPi shields were rev A. If you bought one recently, you have Rev C.
4. The program script can automatically restore your settings and devices after upgrading. If you are uploading to a virgin Arduino, just answer NO to both.
5. Next, just click the program button. If your script was started by CRON, the output will appear in the black box.
6. The BrewPi script will automatically restart after programming and report which version of BrewPi it has found on the Arduino.

Troubleshooting

- If you're prompted with an error: "cannot move uploaded file" rerun the fix permissions script in `sudo sh /home/brewpi/fixPermissions.sh`.
- If saving devices in the device manager does not work, your EEPROM was probably not reset properly. Try reprogramming without settings and devices restore.
- Check whether it is not a hardware problem by programming your Arduino using the Arduino IDE. Just open the *blink* example and click *upload*.

Programming without the web interface

If you are having trouble programming through the web interface, you can try running `programArduinoFirstTime.py`, which is located in your brewpi script directory. First edit it to have the correct settings. Then run it with:

```
sudo python /home/brewpi/programArduinoFirstTime.py
```

Please let us know if you had to resort to this option, it should not be necessary.

CHAPTER 5

Configuring your devices

Since BrewPi 0.2, the hardware setup is dynamic and flexible: you can install and uninstall hardware from the web interface. This is all done from the *Device Configuration* tab in the web interface. From the device manager you can assign hardware (temp sensors, SSRs, etc.) to functions.

The current implementation of the device manager is a bit complicated and basic, but we are working on a more user friendly interface. I hope that this document will help you set it up.

Your devices just after programming

If you have just uploaded a HEX file to your Arduino and the EEPROM was reset, it will depend on your shield version which devices are already installed.

RevC shields

For RevC shields no devices are installed by default.

RevA shields

For RevA shields your devices are set up just like they were in version 0.1 for backwards compatibility. A heater, cooler and door switch are already installed. Also, 2 temperature sensors are installed as 'First device on bus'. You will notice that the 2 temperature sensors are also listed under *Detected devices*, based on their address.

If you would like to switch from 'First device on bus' to address based assignment, remove the 2 installed temperature sensors by setting their function to *None* and clicking apply. After that you can install the detected sensors based on their address. A RevA shield has two OneWire ports. You can connect multiple sensors to both if you use address based sensors.

Receiving the device list

Click *Refresh device list* to receive an updated list of installed and detected devices from the Arduino. To be able to receive the device list, the BrewPi script has to be running.

Installed Devices

All devices that are assigned to a function are found under *Installed devices*.

Detected Devices

The detected devices list shows all devices that are automatically discovered by BrewPi, these include:

- All OneWire devices (temperature sensors and DS2413 OneWire switches)
- All pins for which there is a terminal on your shield

Device properties

Each device has the following properties:

Device setting/property	What it does
Device slot	A device is installed into a device slot. This is a unique number used to identify the device. When configuring your devices, make sure there are no 2 devices with the same slot.
Assigned to (chamber)	Each device is assigned to a Chamber. Currently there is only Chamber 1, but we are preparing for future multi-chamber support. Select *Chamber 1* for all your devices.
Assigned to (beer)	In each chamber, each device is either a *Chamber device* or assigned to a beer. Currently all support devices are chamber devices, except for the beer temperature sensor which should be assigned to *Beer 1*
Function	This is the most important setting for your device. Here you can set what the device should do. The function list is automatically limited to functions that fit with the hardware type. Currently supported functions: * **Chamber temp**: the sensor in the fridge (chamber device). * **Beer temp**: the sensor in your beer (beer device). * **Room temp**: measures any temperature you want, but is not used in the algorithm, just for logging. (chamber device) * **Chamber cooler**: The output that controls your fridge compressor (chamber device) * **Chamber heater**: The output that controls your heater (chamber device) * **Chamber light**: This output is activated when the door is opened (see chamber door) and can also be used as heater by enabling *light as heater* in advanced settings * **Chamber door**: an input that detects when the fridge door is opened.
Device type	Not user configurable, set automatically based on assigned function.
Hardware type	Not user configurable, set automatically. (*Temp Sensor*, *Digital pin* or *OneWire actuator*)

Device type	Not user configurable, set automatically based on assigned function.

Installing new devices and assigning them to a function

You can install a device by changing the properties to a correct configuration and hitting *Apply*. If the values are accepted by the Arduino, your device will show up under *Installed Devices* after refreshing the list.

Please refer to the screenshot below for a reference configuration with all devices installed currently supported by BrewPi. Just leave out any devices you don't have.

Uninstalling a device

To uninstall a device, just set it's function to *None*, hit *Apply* and refresh your device list.

Done!

That was the last step of 'Getting started with Brewpi'! Enjoy using your BrewPi and don't forget to come and have a chat in the BrewPi IRC channel!

CHAPTER 6

Updating BrewPi with our upgrade script

We have written a Python script to help you check for updates and apply them. This script checks both the web interface and the brewpi script directory for updates.

Getting the update script

If you didn't clone the brewpi-tools repository on GitHub repository already, you can do it with the following command. This will clone it from GitHub into your current user's home directory. You can put the brewpi-tools dir anywhere you like, the instructions here assume the pi user's home directory.

```
git clone https://github.com/BrewPi/brewpi-tools.git ~/brewpi-tools
```

Updating BrewPi

Please stop the script before running the updater, using the web interface.

To run the update script, use the following command:

```
sudo ~/brewpi-tools/updater.py
```

The script will display a menu to to choose which branch you would like to update. It defaults to the currently active branch. If you pick another branch, it will check it out for you.

The script will pull from the remote to update. If merging the updates fails, the script will ask you to stash your local changes. This commits your changes to the git stash and bring your repository back to its original state. You can get your changes back with 'git stash pop', but be warned that your changes could be incompatible with the latest updates.

Scripts that run after the update

After updating, the updater calls the utils/runAfterUpdate.sh script from the brewpi-scripts directory. This script will install any new dependencies, update the CRON job and fix file permissions.

Using the updater with your own remote / GitHub account.

The update script just reads the remotes from the config files in the repositories. If you add your own remotes, it will display a menu to choose a remote to update from. So if you forked BrewPi on GitHub, you can also use the updater to update from your own repository.

CHAPTER 7

Frequently Asked Questions (FAQ)

Sorry, not much to see here yet, but I am aware that this section is needed badly! Will fill this with useful content ASAP!

Running Multiple Arduinos on the Same RaspberryPi

In some cases it may be desired to run multiple Arduinos on the same RaspberryPi (RPi). A common reason for doing this is to support multiple chambers (one Arduino per chamber) with a single RPi.

This guide explains how to configure your RPi to run two instances of BrewPi to support two Arduinos controlling two chambers (top and bottom). The use of the *manual installation process* is necessary. The concepts in this guide can also be used to add more than Arduinos, although there will be some limit as to the number of script instances that the RPi can handle.

Setup udev rules for the Arduinos

By using udev, you can create static device node symlinks based on the port of the USB hub that each Arduino is plugged into.[1] This allows you to be confident that each script instance is always controlling the correct Arduino.

Determine the USB hub port identifiers

With only one Arduino connected, issue the following command to see which device node the Arduino is currently using:

```
ls /dev/ttyACM*
```

Then issue the following command, using the device node name found in the previous step:

[1] How to distinguish between identical USB-to-serial adapters? - Ask Ubuntu

```
udevadm info -a -n /dev/arduino_bottom | less
```

You will get a bunch of output. The relevant `looking at parent device` section will be the one that contains a line that states `ATTRS{product}=="Arduino Leonardo"` (your Arduino model may differ). Here is the output from the section to look at from one of my Arduinos:

```
looking at parent device '/devices/platform/bcm2708_usb/usb1/1-1/1-1.3/1-1.3.3/1-1.3.
↪3.3':
  KERNELS=="1-1.3.3.3"
  SUBSYSTEMS=="usb"
  DRIVERS=="usb"
  ATTRS{bDeviceSubClass}=="00"
  ATTRS{bDeviceProtocol}=="00"
  ATTRS{devpath}=="1.3.3.3"
  ATTRS{idVendor}=="2341"
  ATTRS{speed}=="12"
  ATTRS{bNumInterfaces}==" 3"
  ATTRS{bConfigurationValue}=="1"
  ATTRS{bMaxPacketSize0}=="64"
  ATTRS{busnum}=="1"
  ATTRS{devnum}=="9"
  ATTRS{configuration}==""
  ATTRS{bMaxPower}=="500mA"
  ATTRS{authorized}=="1"
  ATTRS{bmAttributes}=="80"
  ATTRS{bNumConfigurations}=="1"
  ATTRS{maxchild}=="0"
  ATTRS{bcdDevice}=="0100"
  ATTRS{avoid_reset_quirk}=="0"
  ATTRS{quirks}=="0x0"
  ATTRS{version}==" 2.00"
  ATTRS{urbnum}=="19"
  ATTRS{ltm_capable}=="no"
  ATTRS{manufacturer}=="Arduino LLC"
  ATTRS{removable}=="unknown"
  ATTRS{idProduct}=="8036"
  ATTRS{bDeviceClass}=="00"
  ATTRS{product}=="Arduino Leonardo"
```

The important line in the output above is the `KERNELS` line. Write that line down and save it for later.

```
KERNELS=="1-1.3.3.3"
```

Next, move the Arduino to the next port on the USB hub that you will use for another Arduino, and repeat the steps previously listed in this section. Repeat the process for as many Arduinos as you will be using.

Write the udev rules

Now that the identifier for each USB hub port has been obtained, the udev rules can be written. In my case, I have the following identifiers:

```
KERNELS=="1-1.3.3.3"
KERNELS=="1-1.3.3.4"
```

Create the file `/etc/udev/rules.d/99-arduino.rules` with contents similar to the following:

```
SUBSYSTEM=="tty", KERNEL=="ttyACM*", KERNELS=="1-1.3.3.3", SYMLINK+="arduino_bottom"
SUBSYSTEM=="tty", KERNEL=="ttyACM*", KERNELS=="1-1.3.3.4", SYMLINK+="arduino_top"
```

The parameters to change on each line are listed in the table below. The other two parameters that aren't listed are there to help prevent symlinks from being created if a device other than an Arduino is plugged into one of the ports in question on the USB hub.

Parameter	Value
KERNELS=="1-1.3.3.3"	Set to the identifier from the previous section that corresponds to the port you're working with.
SYMLINK+="arduino_bottom"	Set to the name of the symlink you wish to create in `/dev/`. Do not include the leading `/dev/`.

In the example above, I end up with the symlinks `/dev/arduino_bottom` and `/dev/arduino_top` when both Arduinos are connected to their respective ports. The symlink names reflect which chamber each Arduino controls.

Once the udev rules file is created, disconnect your Arduino and then reload udev before connecting all of the Ardiunos to their respective ports.

```
sudo /etc/init.d/udev reload
```

Install BrewPi

Install the BrewPi script and web interface manually as described in the *manual installation process*, noting the following changes:

- `git clone` brewpi-script into subdirectories of `/home/brewpi` instead of directly into `/home/brewpi`. I used `/home/brewpi/top` and `/home/brewpi/bottom` to match the chamber each Arduino controls.
- `git clone` brewpi-www into subdirectories of `/var/www` instead of directly into `/var/www`. I used `/var/www/top` and `/var/www/bottom` to match each script installation directory.
- Fix the permissions manually.
  - **UNTESTED** alternative
    - It looks like `utils/fixPermissions.sh` should work when run from each script instance.
    - If you have other content in `/var/www`, you will likely want to update `webPath` in `fixPermissions.sh` to the directory of the corresponding web interface instance.
- Do **not** use `utils/updateCron.sh` or the cron job string in the manual installation instructions. Instead follow the directions in the cron section below.

Modify the config files

Edit the script config files

`settings/config.cfg` needs to be created in each script instance to properly configure them. Here are the config files I'm using.

/home/brewpi/bottom/settings/config.cfg

```
scriptPath = /home/brewpi/bottom/
wwwPath = /var/www/bottom/
port = /dev/arduino_bottom
altport = /dev/null
boardType = leonardo
```

/home/brewpi/top/settings/config.cfg

```
scriptPath = /home/brewpi/top/
wwwPath = /var/www/top/
port = /dev/arduino_top
altport = /dev/null
boardType = leonardo
```

Variable explanation

Variable	Value
scriptPath	Set to the full path of this script instance. Include the trailing slash.
wwwPath	Set to the full path of the web interface instance that corresponds to this script instance. Include the trailing slash.
port	Set to the device node symlink for the Arduino that corresponds to this script instance. This symlink was set up in the udev rules section above.
altport	Set to `/dev/null` so that the use of the default alternate port (/dev/ttyACM1) will not be attempted. Because the device node symlink will always be correct, you don't want an alternate port to be used.
boardType	Set to your Arduino board type.

Edit the web interface config files

`config_user.php` needs to be created in each web interface instance to properly configure them. Here are the config files I'm using.

/var/www/bottom/config_user.php

```
<?php
        // The default settings in config.php are overruled by the settings in config_
↪user.php
        // To use custom settings, copy this file to config_user.php and make your␣
↪changes in config_user.php
        // do not add a php closing tag, because newlines after closing tag might be␣
↪included in the html

        // Do not include a trailing slash on the path
        $scriptPath = '/home/brewpi/bottom';
```

/var/www/top/config_user.php

```
<?php
        // The default settings in config.php are overruled by the settings in config_
↪user.php
        // To use custom settings, copy this file to config_user.php and make your␣
↪changes in config_user.php
        // do not add a php closing tag, because newlines after closing tag might be␣
↪included in the html

        // Do not include a trailing slash on the path
        $scriptPath = '/home/brewpi/top';
```

Variable explanation

Variable	Value
$scriptPath	Set to the full path of the script instance that corresponds to this web interface instance. Do not include a trailing slash.

Set up cron jobs to start the scripts

Create cron job files for each script instance. Here are the config files I'm using.

/etc/cron.d/brewpi_bottom

```
PYTHON=/usr/bin/python
SCRIPTPATH=/home/brewpi/bottom

* * * * * brewpi $PYTHON $SCRIPTPATH/brewpi.py --config $SCRIPTPATH/settings/config.
↪cfg --checkstartuponly --dontrunfile; [ $? != 0 ] && $PYTHON -u $SCRIPTPATH/brewpi.
↪py --config $SCRIPTPATH/settings/config.cfg 1>$SCRIPTPATH/logs/stdout.txt 2>>
↪$SCRIPTPATH/logs/stderr.txt &
```

/etc/cron.d/brewpi_top

```
PYTHON=/usr/bin/python
SCRIPTPATH=/home/brewpi/top

* * * * * brewpi $PYTHON $SCRIPTPATH/brewpi.py --config $SCRIPTPATH/settings/config.
↪cfg --checkstartuponly --dontrunfile; [ $? != 0 ] && $PYTHON -u $SCRIPTPATH/brewpi.
↪py --config $SCRIPTPATH/settings/config.cfg 1>$SCRIPTPATH/logs/stdout.txt 2>>
↪$SCRIPTPATH/logs/stderr.txt &
```

Variable and command explanation

Variable	Value
PYTHON	Set to the full path of the Python binary.
SCRIPTPATH	Set to the full path of the script instance that corresponds to this cron job. Do not include a trailing slash.

`--config $SCRIPTPATH/settings/config.cfg` is specified for both invocations of the script in the cron job so that BrewPi's process monitoring can see that each script instance is unique. For a description of the rest of the items in the cron job command, see the *manual installation process cron job page*.

Updating

I have not investigated whether it is safe to use the updater script from `brewpi-tools`, so at this point I would recommend doing updates manually.

References

CHAPTER 8

Changelog

February 7th, 2014

- Changed how the CRON job is managed. Moved all functionality into updateCron.sh.
- Fixed the CRON entry for wifiChecker.sh

January 20th, 2014

In this release: a new legend for the charts, mash temperatures,

Web interface (brewpi-www) | 0.3.0 -> 0.3.1

Brian Schwinn reworked the legend for the beer charts to show much more info and to be a lot prettier.

New legend for the beer chart

- Legend moved to the right instead of on top of the chart.
- Legend now shows in which state BrewPi was (cooling/heating/waiting etc.) on mouse-over
- Enabling/disabling lines in the chart (click the legend item) is now stored in HTML5 Local Storage, so it is remembered when you reload the page.

Various bug fixes and small edits:

- User documentation removed from brewpi-www repo and moved to separate repository (brewpi-userdocs)
- Beer name removed from maintenance panel (start/stop beer is now done by clicking beer name)
- Added a refresh button if the chart has no data points yet
- Fix for DS2413 devices in device manager
- Better error checking when loading logs

- Removed 'profiles' directory from previous beers drop-down menu

All brewpi-www changes on GitHub

BrewPi Python scripts | 0.3.0 -> 0.3.1

Geo did most work for the BrewPi script this time, he added a script to keep Wifi alive and reworked the update script to also update the Arduino.

WiFi Checker Script

A new script in the utils directory can check whether the WiFi connection is working properly and if not, it resets wlan0 using ifup and ifdown. This script can be installed using 'checkScript.sh install'. This adds it to cron.d/brewpi to run every 10 minutes.

Small changes:

- Moved opening serial into BrewPiUtil, so it can be used by the programming script too.
- Added pin list for DIY shield
- Parsing of the commit SHA in the Arduino version string

All brewpi-script changes on GitHub

Arduino code (brewpi-avr) | 0.2.3.1 -> 0.2.4

The biggest changes in the Arduino code for this release are:

- Shift of the internal temperature format from -64/+64 degree Celsius to -16/+112 degree Celsius. BrewPi can now be used to log your mash temperatures too (Elco). Actuators with PWM for mashing will come in the next release.
- Better handling of temperature sensor disconnects and resets (awesome work by Matthew McGowan).
- Added a NetBeans project to build the Arduino code into a standalone windows application (exe), for testing and simulation (Matthew)
- Added googletest framework for unit testing (Matthew)

Sensor reset detection and smart initialization (Matthew)

- Sensor reset detection: the DS18B20 loads values from non-volatile (EEPROM) memory to volatile memory on startup. By writing a different value to the volatile memory after init, we can see when the sensors have been reset between reads. When a reset is detected, we know not to thrust the next read form the sensor (which defaults to 85C)
- Removed unused fields and code in the DallasTemperature library for reduced code size and memory footprint.
- Smarter re-initialization of sensors and filters. It is now only done after a number of failed reads.

Various changes

- Changed receive function to handle all incoming messages instead of one each main loop. This bug should have been noticed much earlier. It could cause the Arduino to be slow to respond to messages in certain circumstances.
- Included Arduino files in the project and removed USB HID driver support from Arduino files. This saves 918 bytes!
- Changed how the processor/Arduino type is detected in the build (now based on processor)
- Removed devices installed by default for RevA shields. These caused conflicts when devices were restored too. Now RevA behaves like RevC.

- Support for using DS2413 as switch input (Matthew)
- Added a minimum for overshoot estimators. They could not recover from being zero before this fix.
- Include commit SHA in version number

All brewpi-www changes on GitHub

Install and update tools (Brewpi-tools)

- Update script now also checks Arduino version and can reprogram the Arduino. It downloads the latest hex file from the BrewPi server.
- Added Wifi check script to install (add to cron.d)

All brewpi-tools changes on GitHub

December 23, 2013

Arduduino code(brewpi-avr) | 0.2.3 -> 0.2.3.1

Hotfix: pidMax was printed as a temperature (with offset in Fahrenheit) which caused it to be restored incorrectly when programming the Arduino.

October 22, 2013

It's been a long time since a master release, but in this release we made some major steps to make releasing easier in the future.

The biggest changes in this release are:

- Added a native interface for beer profiles. No more google spreadsheets! Huge thank you to Brian Schwinn (bschwinn) for his hard work on this feature.
- Added an easy menu to start a new brew, start/stop/pause data logging
- Added an install and update script! Just run the script and afterwards go straight to flashing your Arduino and setting up devices. Huge thank you to Geo van O. for his hard work on these scripts.
- Tweaked the temperature control algorithm to reduce overshoot.
- Use cron.d instead of crontab to make automated updating of the cron job easier

Instructions for installing/updating BrewPi can be found in the documentation. and the scripts are part of the new brewpi-tools repository on GitHub.

Detailed changes per repository are displayed below.

Web interface (brewpi-www) | 0.2 -> 0.3

- New interface to create/edit/save/load profiles
- Added dialog to start/stop/pause data logging and to start a new brew
- Split config files in default config in source control and user config outside of source control
- Better way to hide page elements while rendering

- Room temp and fridge temp are now hidden by default to reduce clutter. Click the circles next to the graph to show them.
- Bug fixes and layout fixes

All brewpi-www changes on GitHub

BrewPi Python scripts | 0.2 -> 0.3

- **Changed the way python works with profiles:**
  - Support for disabling temperature control in the profile
  - Internal change to the temperature profile format to work with the new interface
- Default first beer is now 'My First BrewPi Run', so it does not append to the sample data
- **Added a utils directory with scripts to:**
  - Fix permissions
  - Update the CRON job for this version
  - Install all dependencies
  - All of the above: runAfterUpdate.sh
- Resolved startup issues with bootloaders that take longer (stuck at script starting up)
- Added altport setting in config: script will try this alternative port (ttyACM1) wen default port cannot be found
- When restoring settings to the Arduino, send them in a specific order. Fahrenheit settings could be interpreted as Celcius before.
- Commands to start/stop/pause logging (script side)
- Better error exception and lots of bug fixes

All brewpi-script changes on GitHub

Arduino code (brewpi-avr) | 0.2.0 -> 0.2.3

- **Algorithm changes to prevent overshooting beer temperature in fast fridges**
  - Immediately stop heating/cooling when beer hits target, regardless of fridge temp/target
  - Reduced minimum on time to 3 minutes (10 minutes for fridge constant cooling to prevent fast cycling)
  - Increased update rate of slope filter, so it has less delay
  - Reduced default PID parameters. A slow controller is better than overshoot
  - Integrator is only updated in IDLE
  - Added tiny idle zone for beer temp (-0.5/+0.5 LSB before filtering)
- Added PID max setting: the max difference between generated fridge setting and beer setting
- Output a new data point on every state transition
- Do not go into heating when no heater is installed
- Time on display is now printed as hours, minutes, seconds: 01h01m39 or 01m39
- Fix for actuators being active for 1 second at boot

- Changed when data is written to EEPROM to reduce number of writes
- Enabled internal pull-ups, so the shield also works well without the display backpack connected.
- Inverted pin mode is now default for new devices
- Beep as first thing at boot, so you know when the bootloader ends and brewpi starts

All brewpi-avr changes on GitHub

BrewPi tools for bootstrapping and updating (brewpi-tools) | New

- **Install script that performs most steps that you previously had to do manually on the command line, mainly:**
  - Creating users, directories, etc
  - Installing dependencies (web server, python libraries, etc)
  - Cloning the repositories
  - Setting up the CRON job
- **Update script to make it easy to check for updates and apply them**
  - Check your configured remotes for updates (not just the official repository)
  - Pull updates from GitHub
  - Ask to stash changes on merge conflicts
  - Switch branches

All brewpi-tools changes on GitHub