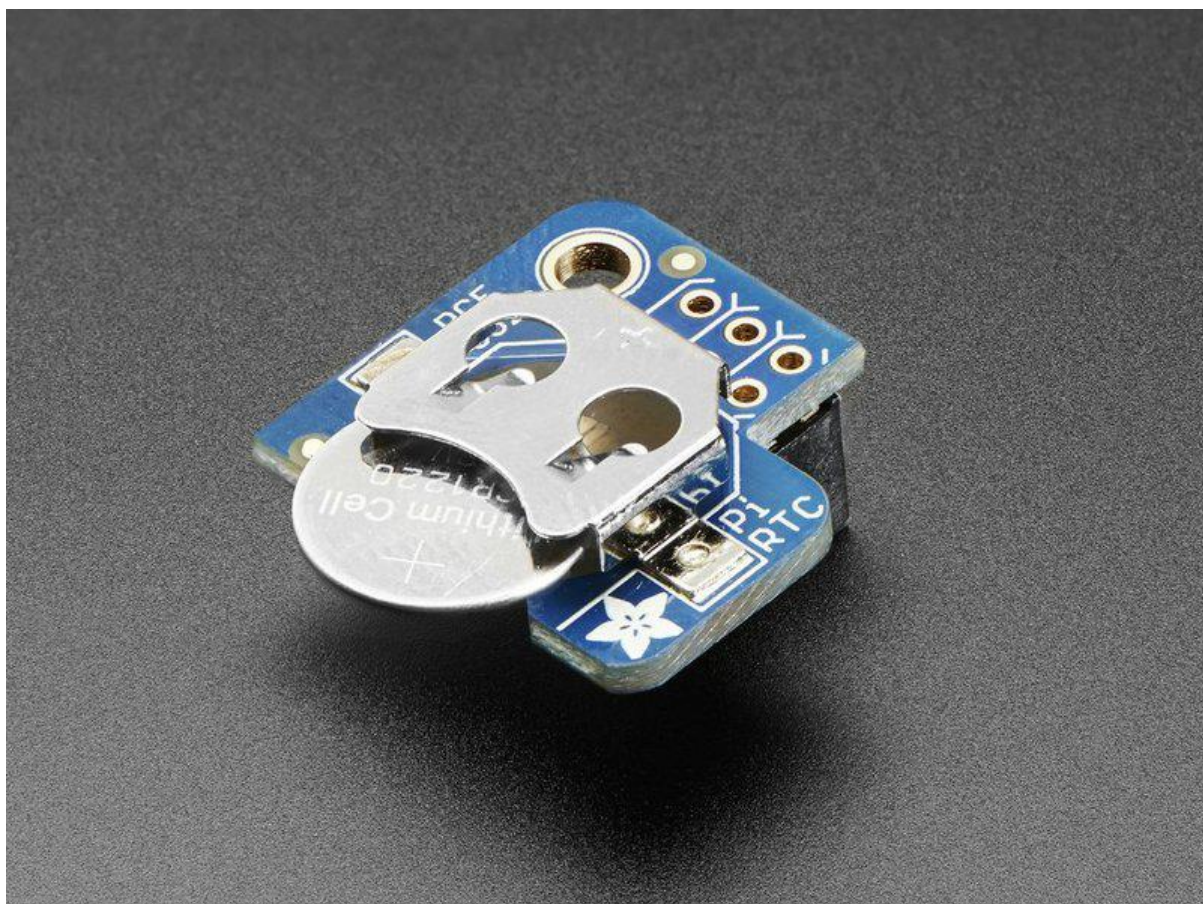




Adding a Real Time Clock to Raspberry Pi

Created by lady ada



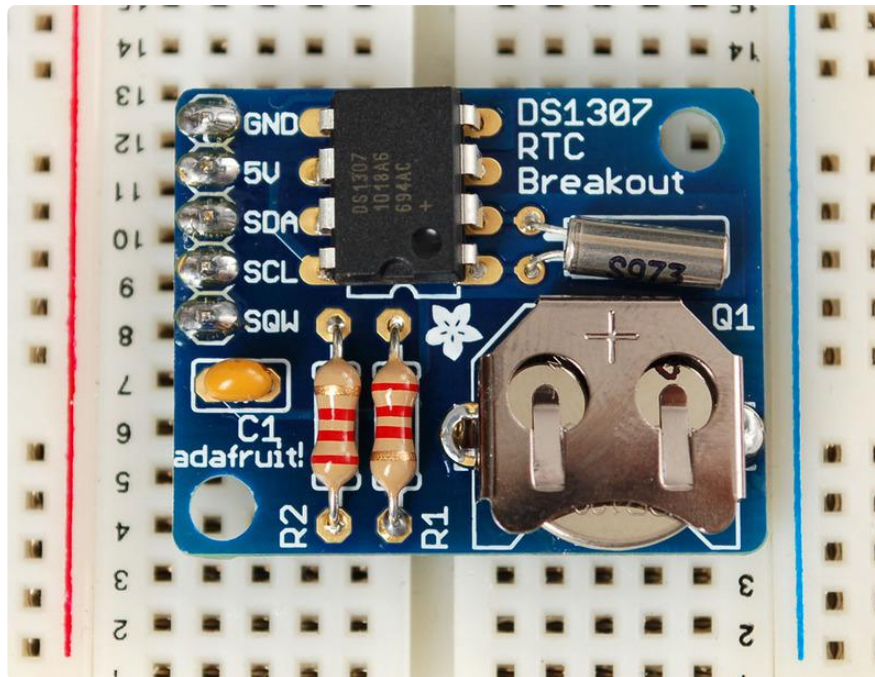
<https://learn.adafruit.com/adding-a-real-time-clock-to-raspberry-pi>

Last updated on 2021-11-15 05:51:19 PM EST

Table of Contents

| | |
|--|----|
| Overview | 3 |
| Wiring the RTC | 3 |
| Set Up & Test I2C | 6 |
| • Set up I2C on your Pi | 6 |
| • Verify Wiring (I2C scan) | 6 |
| Set RTC Time | 7 |
| • Raspberry Pi OS's with systemd | 7 |
| • Sync time from Pi to RTC | 10 |
| • Raspbian Wheezy or other pre-systemd Linux | 11 |

Overview



This tutorial requires a Raspberry Pi running a kernel with the RTC module and DS1307 module included. Current Raspbian distros have this, but others may not!

The Raspberry Pi is designed to be an ultra-low cost computer, so a lot of things we are used to on a computer have been left out. For example, your laptop and computer have a little coin-battery-powered 'Real Time Clock' (RTC) module, which keeps time even when the power is off, or the battery removed. To keep costs low and the size small, an RTC is not included with the Raspberry Pi. Instead, the Pi is intended to be connected to the Internet via Ethernet or WiFi, updating the time automatically from the global ntp (network time protocol) servers

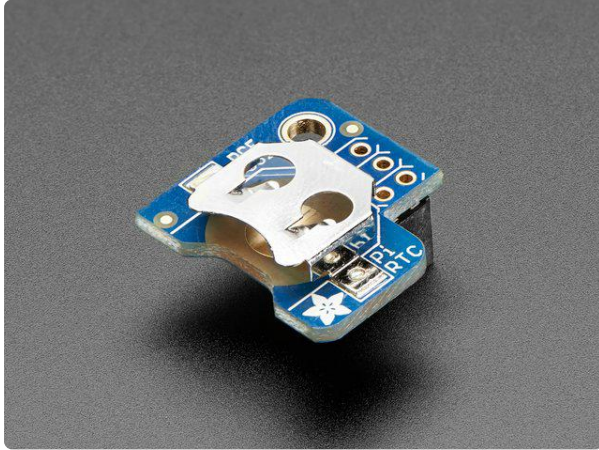
For stand-alone projects with no network connection, you will not be able to keep the time when the power goes out. So in this project we will show you how to add a low cost battery-backed RTC to your Pi to keep time!

Wiring the RTC

To keep costs low, the Raspberry Pi does not include a Real Time Clock module. Instead, users are expected to have it always connected to WiFi or Ethernet and keep time by checking the network. Since we want to include an external module, we'll have to wire one up.

We have three different RTC we suggest, PCF8523 is inexpensive, DS1307 is most common, and DS3231 is most precise. Any of them will do!

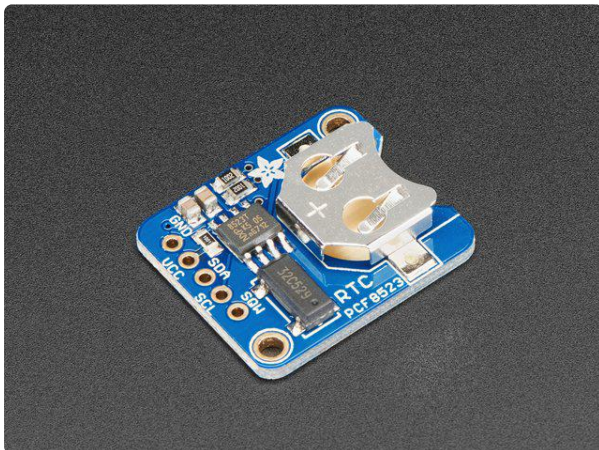
While the DS1307 is historically the most common, its not the best RTC chipset, we've found!



Adafruit PiRTC - PCF8523 Real Time Clock for Raspberry Pi

This is a great battery-backed real time clock (RTC) that allows your Raspberry Pi project to keep track of time if the power is lost. Perfect for data-logging, clock-building,...

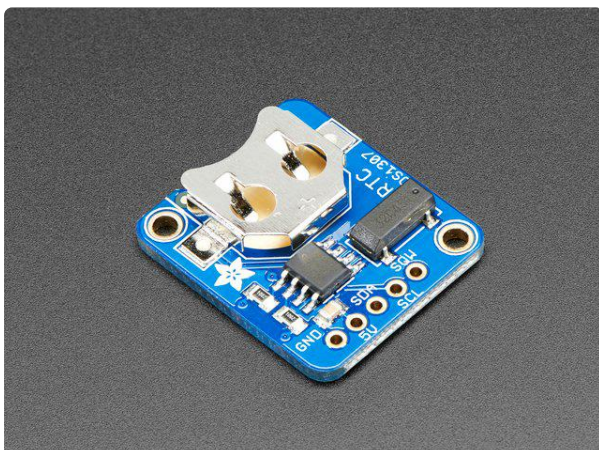
<https://www.adafruit.com/product/3386>



Adafruit PCF8523 Real Time Clock Assembled Breakout Board

This is a great battery-backed real time clock (RTC) that allows your microcontroller project to keep track of time even if it is reprogrammed, or if the power is lost. Perfect for...

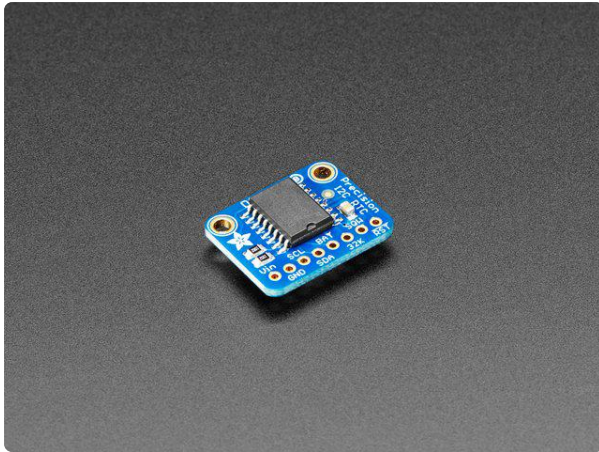
<https://www.adafruit.com/product/3295>



Adafruit DS1307 Real Time Clock Assembled Breakout Board

This is a great battery-backed real time clock (RTC) that allows your microcontroller project to keep track of time even if it is reprogrammed, or if the power is lost. Perfect for...

<https://www.adafruit.com/product/3296>



Adafruit DS3231 Precision RTC Breakout

The datasheet for the DS3231 explains that this part is an "Extremely Accurate I²C-Integrated RTC/TCXO/Crystal". And, hey, it does exactly what it says...

<https://www.adafruit.com/product/3013>

Don't forget to also install a CR1220 coin cell. In particular the DS1307 won't work at all without it and none of the RTCs will keep time when the Pi is off and no coin battery is in place.



CR1220 12mm Diameter - 3V Lithium Coin Cell Battery

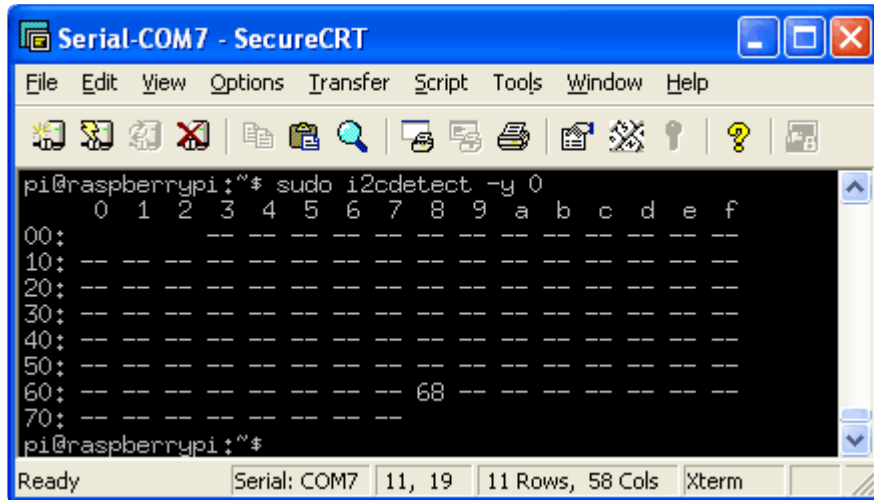
These are the highest quality & capacity batteries, the same as shipped with the iCufflinks, iNecklace, Datalogging and GPS Shields, GPS HAT, etc. One battery per order...

<https://www.adafruit.com/product/380>

Wiring is simple:

1. Connect VCC on the breakout board to the 5.0V pin of the Pi (if using DS1307)
Connect VCC on the breakout board to the 3.3V pin of the Pi (if using PCF8523 or DS3231)
2. Connect GND on the breakout board to the GND pin of the Pi
3. Connect SDA on the breakout board to the SDA pin of the Pi
4. Connect SCL on the breakout board to the SCL pin of the Pi

If you have a much older Pi 1, you will have to run `sudo i2cdetect -y 0` as the I2C bus address changed from 0 to 1



```
pi@raspberrypi:~$ sudo i2cdetect -y 0
   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
20:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
60:  --  --  --  --  --  --  68  --  --  --  --  --  --  --  --
70:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
pi@raspberrypi:~$
```

Once you have the Kernel driver running, i2cdetect will skip over 0x68 and display UU instead, this means its working!

Set RTC Time

Now that we have the module wired up and verified that you can see the module with i2cdetect, we can set up the module.

Don't forget to set up I2C in the previous step!

Raspberry Pi OS's with systemd

This should be the case for any current release. For much older releases without systemd, skip to the next section.

[Thanks to kd8twg for the hints! \(https://adafru.it/ne3\)](https://adafru.it/ne3)

You can add support for the RTC by adding a device tree overlay. Run

```
sudo nano /boot/config.txt
```

to edit the pi configuration and add whichever matches your RTC chip:

```
dtoverlay=i2c-rtc,ds1307
```

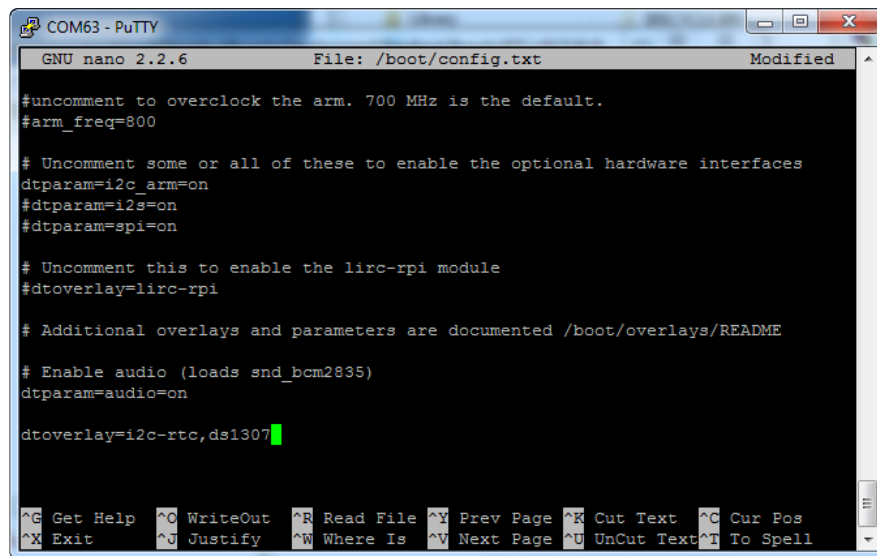

or

```
dtoverlay=i2c-rtc,pcf8523
```

or

```
dtoverlay=i2c-rtc,ds3231
```

to the end of the file



```
COM63 - PuTTY
GNU nano 2.2.6      File: /boot/config.txt      Modified

#uncomment to overclock the arm. 700 MHz is the default.
#arm_freq=800

# Uncomment some or all of these to enable the optional hardware interfaces
dtparam=i2c_arm=on
#dtparam=i2s=on
#dtparam=spi=on

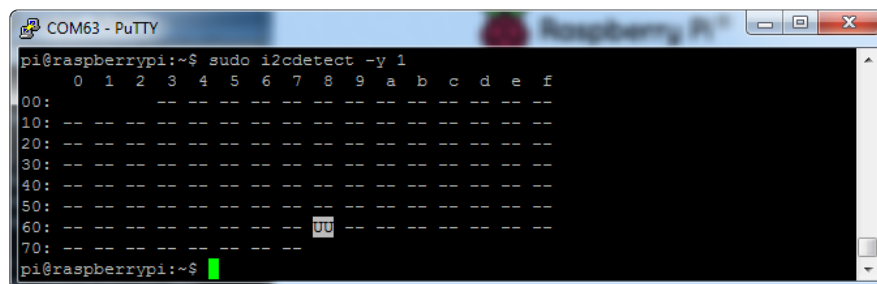
# Uncomment this to enable the lirc-rpi module
#dtoverlay=lirc-rpi

# Additional overlays and parameters are documented /boot/overlays/README

# Enable audio (loads snd_bcm2835)
dtparam=audio=on

dtoverlay=i2c-rtc,ds1307
```

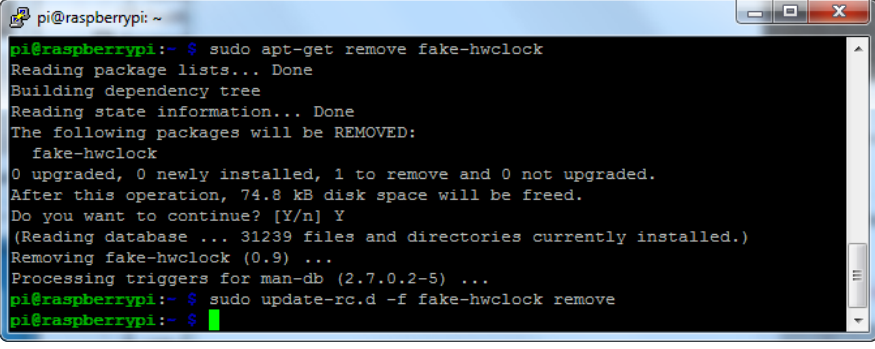
Save it and run **sudo reboot** to start again. Log in and run **sudo i2cdetect -y 1** to see the UU show up where 0x68 should be



```
COM63 - PuTTY
pi@raspberrypi:~$ sudo i2cdetect -y 1
 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- UU -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
pi@raspberrypi:~$
```

Disable the "fake hwclock" which interferes with the 'real' hwclock

- **sudo apt-get -y remove fake-hwclock**
- **sudo update-rc.d -f fake-hwclock remove**
- **sudo systemctl disable fake-hwclock**

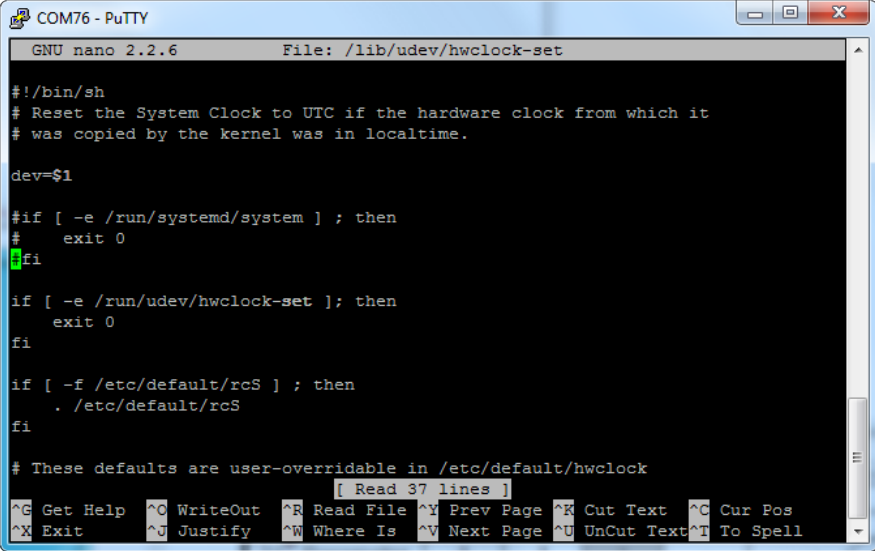


```
pi@raspberrypi:~  
pi@raspberrypi:~$ sudo apt-get remove fake-hwclock  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following packages will be REMOVED:  
  fake-hwclock  
0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded.  
After this operation, 74.8 kB disk space will be freed.  
Do you want to continue? [Y/n] Y  
(Reading database ... 31239 files and directories currently installed.)  
Removing fake-hwclock (0.9) ...  
Processing triggers for man-db (2.7.0.2-5) ...  
pi@raspberrypi:~$ sudo update-rc.d -f fake-hwclock remove  
pi@raspberrypi:~$
```

Now with the fake-hw clock off, you can start the original 'hardware clock' script.

Run `sudo nano /lib/udev/hwclock-set` and comment out these three lines:

```
#if [ -e /run/systemd/system ] ; then  
# exit 0  
#fi
```



```
COM76 - PuTTY  
GNU nano 2.2.6      File: /lib/udev/hwclock-set  
  
#!/bin/sh  
# Reset the System Clock to UTC if the hardware clock from which it  
# was copied by the kernel was in localtime.  
  
dev=$1  
  
#if [ -e /run/systemd/system ] ; then  
#   exit 0  
#fi  
  
if [ -e /run/udev/hwclock-set ] ; then  
  exit 0  
fi  
  
if [ -f /etc/default/rcS ] ; then  
  . /etc/default/rcS  
fi  
  
# These defaults are user-overridable in /etc/default/hwclock  
[ Read 37 lines ]  
^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
```

Also comment out the two lines

```
/sbin/hwclock --rtc=$dev --systz --badyear
```

and

```
/sbin/hwclock --rtc=$dev --systz
```

```

if [ yes = "$BADYEAR" ] . then
# /sbin/hwclock --rtc=$dev --systz --badyear
/sbin/hwclock --rtc=$dev --hctosys --badyear
else
# /sbin/hwclock --rtc=$dev --systz
/sbin/hwclock --rtc=$dev --hctosys
fi

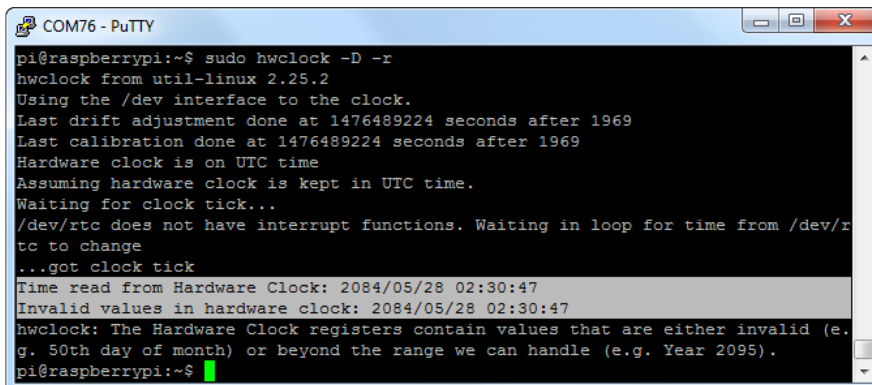
# Note 'touch' may not be available in initramfs
> /run/udev/hwclock-set

```

Sync time from Pi to RTC

When you first plug in the RTC module, it's going to have the wrong time because it has to be set once. You can always read the time directly from the RTC with **sudo hwclock -r**

(ignore use of deprecated -D parameter)

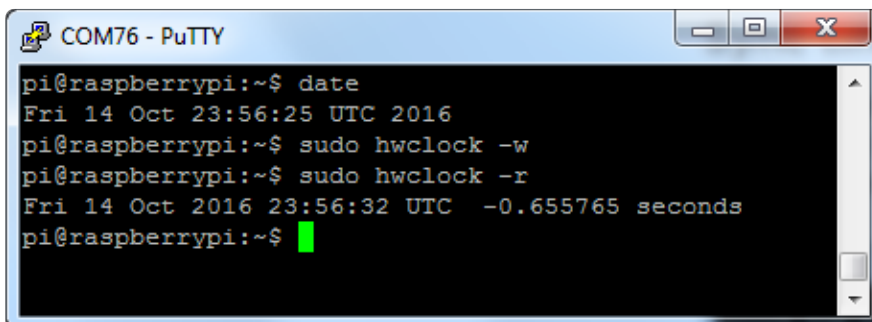


```

pi@raspberrypi:~$ sudo hwclock -D -r
hwclock from util-linux 2.25.2
Using the /dev interface to the clock.
Last drift adjustment done at 1476489224 seconds after 1969
Last calibration done at 1476489224 seconds after 1969
Hardware clock is on UTC time
Assuming hardware clock is kept in UTC time.
Waiting for clock tick...
/dev/rtc does not have interrupt functions. Waiting in loop for time from /dev/rtc to change
...got clock tick
Time read from Hardware Clock: 2084/05/28 02:30:47
Invalid values in hardware clock: 2084/05/28 02:30:47
hwclock: The Hardware Clock registers contain values that are either invalid (e.g. 50th day of month) or beyond the range we can handle (e.g. Year 2095).
pi@raspberrypi:~$

```

You can see, the date at first is invalid! You can set the correct time easily. First run **date** to verify the time is correct. Plug in Ethernet or WiFi to let the Pi sync the right time from the Internet. Once that's done, run **sudo hwclock -w** to write the time, and another **sudo hwclock -r** to read the time



```

pi@raspberrypi:~$ date
Fri 14 Oct 23:56:25 UTC 2016
pi@raspberrypi:~$ sudo hwclock -w
pi@raspberrypi:~$ sudo hwclock -r
Fri 14 Oct 2016 23:56:32 UTC -0.655765 seconds
pi@raspberrypi:~$

```

Once the time is set, make sure the coin cell battery is inserted so that the time is saved. You only have to set the time once

That's it! Next time you boot the time will automatically be synced from the RTC module

hwclock: ioctl(RTC_RD_TIME) to /dev/rtc0 to read the time failed: Invalid argument

If you are getting an error message like this when trying to read/write to the RTC, make sure you have a good coin cell battery installed.

Raspbian Wheezy or other pre-systemd Linux

First, load up the RTC module by running

```
sudo modprobe i2c-bcm2708
sudo modprobe i2c-dev
sudo modprobe rtc-ds1307
```

Then, as root (type in `sudo bash`) run

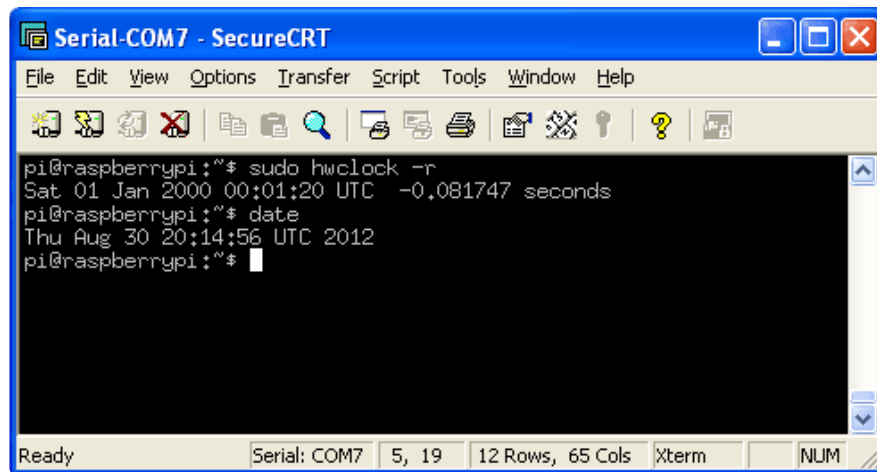
```
echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
```

If you happen to have an old Rev 1 Pi, type in

```
echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-0/new_device
```

You can then type in `exit` to drop out of the root shell.

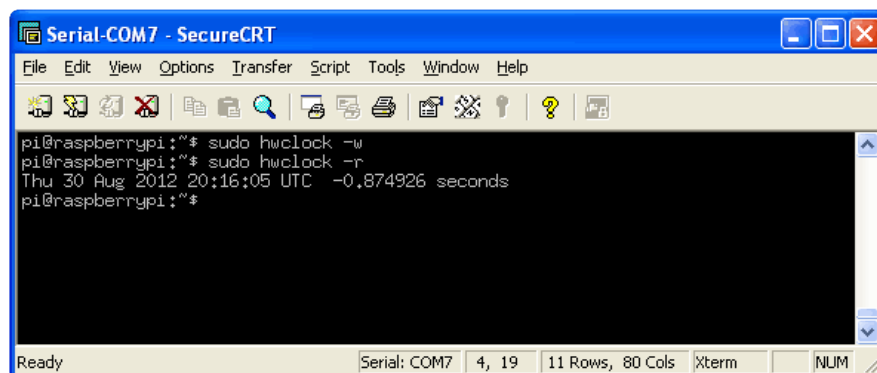
Then check the time with `sudo hwclock -r` which will read the time from the DS1307 module. If this is the first time the module has been used, it will report back Jan 1 2000, and you'll need to set the time



```
pi@raspberrypi:~$ sudo hwclock -r
Sat 01 Jan 2000 00:01:20 UTC -0.081747 seconds
pi@raspberrypi:~$ date
Thu Aug 30 20:14:56 UTC 2012
pi@raspberrypi:~$
```

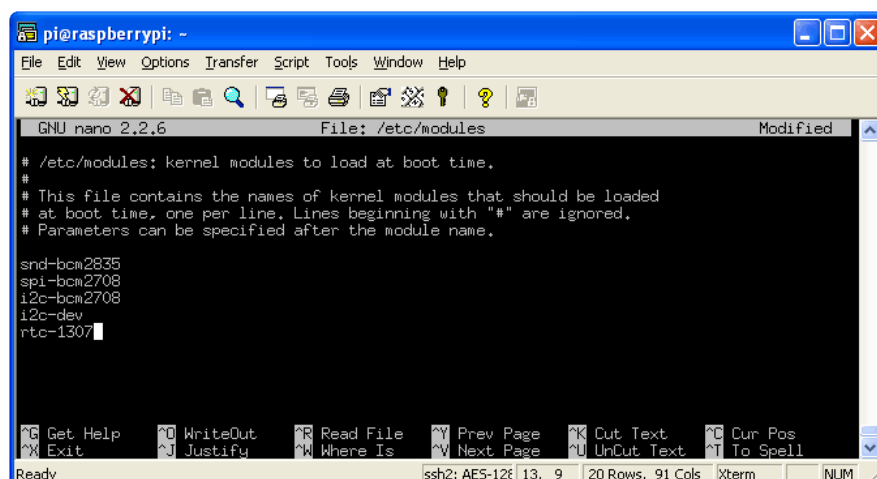
First you'll need to get the right time set on the Pi, the easiest way is to connect it up to Ethernet or Wifi - it will automatically set the time from the network. Once the time is correct (check with the date command), run `sudo hwclock -w` to write the system time to the RTC

You can then verify it with `sudo hwclock -r`



```
pi@raspberrypi:~$ sudo hwclock -w
pi@raspberrypi:~$ sudo hwclock -r
Thu 30 Aug 2012 20:16:05 UTC -0.874926 seconds
pi@raspberrypi:~$
```

Next, you'll want to add the RTC kernel module to the `/etc/modules` list, so its loaded when the machine boots. Run `sudo nano /etc/modules` and add `rtc-ds1307` at the end of the file (the image below says `rtc-1307` but its a typo)



```
GNU nano 2.2.6 File: /etc/modules Modified
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
# Parameters can be specified after the module name.

snd-bcm2835
spi-bcm2708
i2c-bcm2708
i2c-dev
rtc-1307
```


Older pre-Jessie raspbian is a little different. First up, you'll want to create the DS1307 device creation at boot, edit /etc/rc.local by running

```
sudo nano /etc/rc.local
```

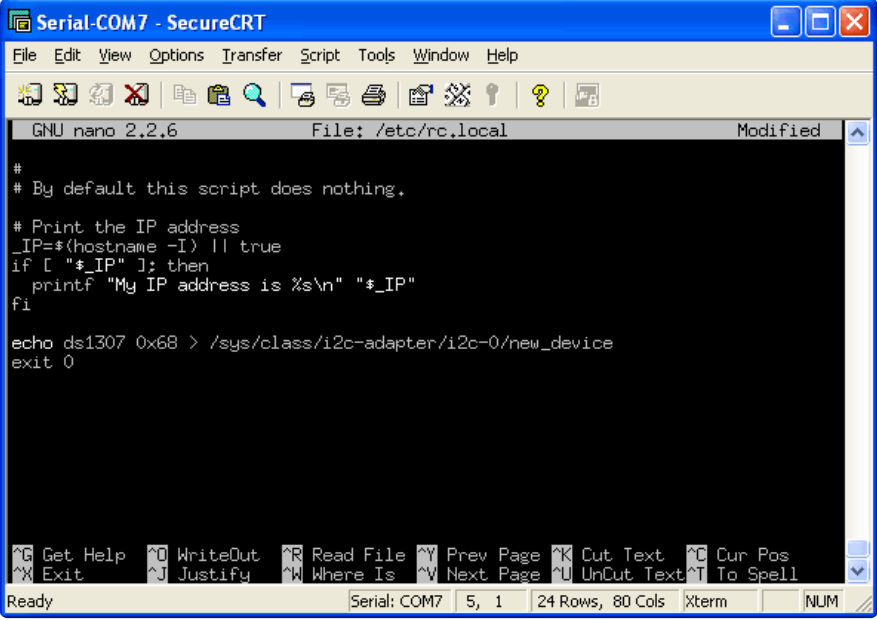
and add:

```
echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-0/new_device (f  
or v1 raspberry pi)
```

```
echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device (f  
or v2 raspberry pi)
```

```
sudo hwclock -s (both versions)
```

before `exit 0` (we forgot the hwclock -s part in the screenshot below)



The screenshot shows a SecureCRT window titled 'Serial-COM7 - SecureCRT'. The terminal displays the contents of the file /etc/rc.local, edited with GNU nano 2.2.6. The file content is as follows:

```
#  
# By default this script does nothing.  
  
# Print the IP address  
_IP=$(hostname -I) || true  
if [ "$_IP" ]; then  
    printf "My IP address is %s\n" "$_IP"  
fi  
  
echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-0/new_device  
exit 0
```

The bottom status bar of the terminal shows 'Ready', 'Serial: COM7', '5, 1', '24 Rows, 80 Cols', 'Xterm', and 'NUM'.

That's it! Next time you boot the time will automatically be synced from the RTC module

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

[Adafruit:](#)

[3386](#)