RPi Documentation

Release 0.1

Yves J. Hilpisch

December 25, 2014

CONTENTS

This Web site and tutorial is about setting up and using the Raspberry Pi for some serious things. Among others, it covers so far:

- using the RPi via **SSH access** and a **fixed IP address**
- using the RPi as **FTP server**
- doing Python-based **data analytics** with the RPi
- building and deploying **Web apps** on the RPi

I assume that you have bought a RPi and all the equipment necessary to use it (power plug, etc.). It is recommended to start using the RPi connected to the Web via an Ethernet cable (for convenience and speed).

Several topics and projects of this tutorial are also interesting for those not (yet) having a RPi but having available or being willing to rent a (small) **cloud server instance** e.g. from **DigitalOcean**. These start at **5 USD per month** for a 1 core, 512 MB, 20 GB SSD configuration. Just follow this link to **register**: https://www.digitalocean.com/?refcode=fbe512dd3dac.

CHAPTER

ONE

SETTING UP THE RPI

The first step is to build a bootable SD card for the RPi. We will use a **Raspbian Debian Wheezy** image in the following which you can download here: http://www.raspberrypi.org/downloads/.

Using a Mac, you can do the following to write the downloaded image to the SD card. First, insert the SD card. On the shell type:

```
diskutil list
```

Using Linux (eg Ubuntu), type:

```
df -h
```

This gives you a list of all disk drives. Identify the SD card with a name like `diskX`.

Then unmount the SD card on the Mac as follows:

```
diskutil unmountDisk /dev/diskX
```

Under Linux do:

```
umount /dev/diskX
```

The next step is to write the OS image to the SD card:

```
sudo dd bs=1m if=os-image.img of=/dev/diskX
```

Here, replace the image name and the disk name with those that apply for you.

CHAPTER

TWO

BOOTING THE RPI

You should connect a monitor via a HDMI cable to the RPi and a keyboard via USB (I am using a keyboard and mouse, both connected via the same USB token). Put the SD card into the RPi and connect it to the power plug. It should now boot.

You will be directed to an options screen where you can do different things, like for example:

- expand the file system to use the full capacity of your SD card (which you should do)
- change the root/pi password (which I assume in the following you do not do)
- enable SSH access (via Advanced Options, which you should do)

After finishing the options setting procedure, the RPi has to re-boot. Once rebooted, you should login as user `pi` with password `raspberry` (if not changed before). The type:

```
sudo apt-get update
```

This might take a while. After that, upgrade you system with:

```
sudo apt-get upgrade
```

You can further use the RPi with a monitor and keyboard connected or you can use it via `ssh` access as one of the small projects explains.

CHAPTER

THREE

SMALL PROJECTS WITH THE RPI

Having set up the RPi, you can now move on and implement one or more of the following smaller projects. Ideally, you should **follow them in the sequence as listed below** since later projects assume (at least to some extent) that you have successfully finished earlier ones.

The documentation is structured as follows:

3.1 Basic Configurations

This section is about some basic, useful configurations of the RPi.

3.1.1 SSH Access

When the RPi boots, being connected via Ethernet cable to a router, it shows at the end of the boot output the **IP address** under which it is connected. Alternatively, for example, using your admin access to your router, identify the IP address under which the RPi is connected. Then using another computer connected to the local network, open the shell and `ssh` to the RPi as follows:

```
ssh pi@192.168.178.xx
```

Here, replace the IP address that applies for you. After you are prompted for it, type in the password `raspberry`. You should now be connected with the RPi.

Fixed IP Address

A **fixed IP address** for the RPi is helpful for a number of reasons. I have configured the router such that the RPi always gets the same IP address, say 192.168.178.xx. For the RPi type the following to **edit the respective settings**:

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

Replace the following line:

```
iface eth0 inet dhcp
```

by these lines:

```
iface eth0 inet static
address 192.168.178.xx
netmask 255.255.255.0
gateway 192.168.178.1
```

Exit the editor with `ctrl+x` saving the changes. Finally, **restart** the network service by:

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

Henceforth, the RPi should be reachable under 192.168.178.xx.

Public IP Address

Using a service like http://whatismyipaddress.com/, you can figure out what your **public router IP address** is (depending on your router this might change regularly; modern ones at least keep their addresses until the next re-boot).

Say, you have figured out that the address is xx.yy.zz.100. Configuring your router to make the IP address of your RPi 192.168.178.xx an "exposed host", should enable you to access the RPi from anywhere via:

```
ssh pi@xx.yy.zz.100
```

Knowing your public IP, you can also set an **A record** (cf. http://support.dnsimple.com/articles/a-record/) of one of your domains to access the RPi, like `http://rpi.mydomain.net` (which should have an A record equal to the public IP xx.yy.zz.100).

Using Public Keys

Connecting to the RPi via `ssh` generally asks for a password. However, you can also generate **RSA keys** and copy the public key to the RPi which then allows you the `ssh` to the RPi without typing your password again and again (when using the same machine and/or keys). Generate keys (if you haven't yet) via:

```
ssh-keygen -t rsa
```

Then **copy the public key** via (Linux):

```
ssh-copy-id pi@192.168.178.xx
```

On the Mac see, for example, https://github.com/beautifulcode/ssh-copy-id-for-OSX.

3.1.2 Other Useful Tools & Configurations

This section provides a collection of other useful tools to work with the RPi (and other Linux/Debian-based servers). It also shows some helpful configurations.

Session Management with Screen

A really helpful tool when working via `ssh` access is **Screen** (cf. http://ss64.com/bash/screen.html). It allows you to have **multiple shells** running at the same time, to disconnect from them and later reconnect. Install it via:

```
sudo apt-get install screen
```

To start it, type:

```
screen
```

A new shell is created by `ctrl+a` then `c`. You switch to the next shell with `ctrl+a` and then `n` (more commands are found under the above link).

Later when you reconnect via `ssh` to the RPi you can **reattach to your Screen session** via:

```
screen -r
```

System Monitoring with htop

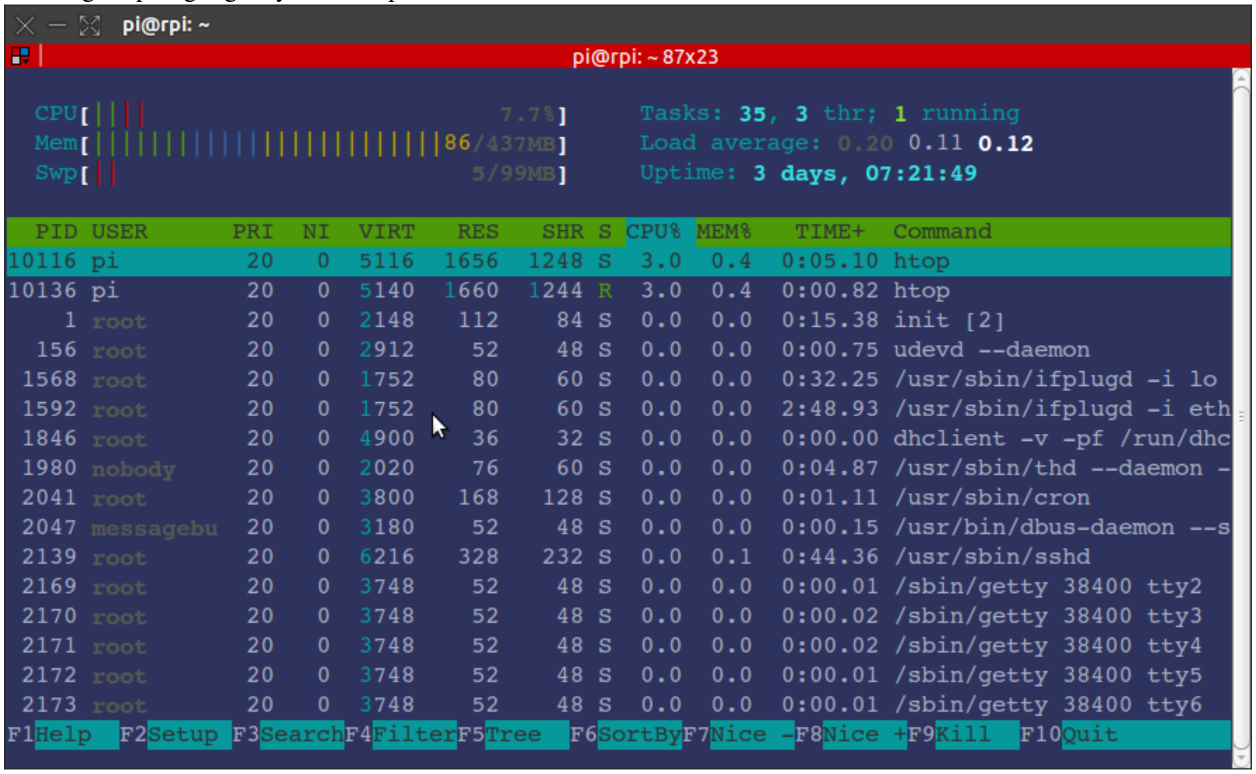
Of course, there a many **useful tools** available in the Linux/RPi world. One you should install as well is **htop** (cf. http://manpages.ubuntu.com/manpages/oneiric/man1/htop.1.html)

You can install it via:

```
sudo apt-get install htop
```

Running htop gives you an overview over the **usage of the system resources and the processes running** (including their memory consumption and CPU usage).

Running htop might give you an output like this one:



```
  CPU[|||                            7.7%]     Tasks: 35, 3 thr; 1 running
  Mem[|||||||||||||||||||||||||86/437MB]     Load average: 0.20 0.11 0.12
  Swp[||                           5/99MB]     Uptime: 3 days, 07:21:49

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
10116 pi         20   0  5116  1656  1248 S  3.0  0.4  0:05.10 htop
10136 pi         20   0  5140  1660  1244 R  3.0  0.4  0:00.82 htop
    1 root       20   0  2148   112    84 S  0.0  0.0  0:15.38 init [2]
  156 root       20   0  2912    52    48 S  0.0  0.0  0:00.75 udevd --daemon
 1568 root       20   0  1752    80    60 S  0.0  0.0  0:32.25 /usr/sbin/ifplugd -i lo
 1592 root       20   0  1752    80    60 S  0.0  0.0  2:48.93 /usr/sbin/ifplugd -i eth
 1846 root       20   0  4900    36    32 S  0.0  0.0  0:00.00 dhclient -v -pf /run/dhc
 1980 nobody     20   0  2020    76    60 S  0.0  0.0  0:04.87 /usr/sbin/thd --daemon -
 2041 root       20   0  3800   168   128 S  0.0  0.0  0:01.11 /usr/sbin/cron
 2047 messagebu  20   0  3180    52    48 S  0.0  0.0  0:00.15 /usr/bin/dbus-daemon --s
 2139 root       20   0  6216   328   232 S  0.0  0.1  0:44.36 /usr/sbin/sshd
 2169 root       20   0  3748    52    48 S  0.0  0.0  0:00.01 /sbin/getty 38400 tty2
 2170 root       20   0  3748    52    48 S  0.0  0.0  0:00.02 /sbin/getty 38400 tty3
 2171 root       20   0  3748    52    48 S  0.0  0.0  0:00.02 /sbin/getty 38400 tty4
 2172 root       20   0  3748    52    48 S  0.0  0.0  0:00.01 /sbin/getty 38400 tty5
 2173 root       20   0  3748    52    48 S  0.0  0.0  0:00.01 /sbin/getty 38400 tty6
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Enlarging the Swap Capacity

The typical RPi comes with 512 MB of memory only. This might not be enough in certain scenarios. One way to increase the memory (if only virtually) is to work with a larger than normal **swap partition**. The following steps generate a **1 GB** swap partition for the RPi.

First, make a **swap directory and swap file**:

```
sudo mkdir /media/swap
sudo dd if=/dev/zero of=/media/swap/swapfile.img bs=1024 count=1M
```

Second, generate the **swap filesystem**:

> sudo mkswap /media/swap/swapfile.img

Third, **edit** the following file:

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

Add this line to that file:

```
/media/swap/swapfile.img swap swap sw 0 0
```

Fourth, **activate** the swap file:

```
sudo swapon /media/swap/swapfile.img
```

Finally, you can **check** whether the new swap configuration is indeed active—either via htop or as follws:

```
pi@rpi ~ $ cat /proc/swaps
Filename                                Type            Size    Used    Priority
/var/swap                               file            102396  6064    -1
/media/swap/swapfile.img                file            1048572 0       -2
pi@rpi ~ $
```

3.2 Raspberry Pi as FTP Server

Setting up the RPi as an FTP server—over which you and you alone have full control—is quite a simple task. First, install the `ProFTP` software through:

```
sudo apt-get install proftpd
```

Configure it to be "standalone". Second, **edit the config file** of the program as follows:

```
sudo nano /etc/proftpd/proftpd.conf
```

Add the following **parameters**:

```
DefaultRoot         ~
AuthOrder           mod_auth_file.c  mod_auth_unix.c
AuthUserFile        /etc/proftpd/ftpd.passwd
AuthPAM             off
RequireValidShell   off
```

Restart the service by:

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

Third, **generate a new user** as follows:

```
sudo adduser ftp --home /home/ftp --shell /bin/bash
```

Change `ftp` and `/home/ftp` to a user name and directory of your liking. Choose a password for the new user.

Using a **FTP client**, you can now connect to your RPi and, for example, store files on it. You can also `ssh` connect to the RPi using the new user credentials.

Generally SD cards hosting the OS of the RPi and serving as file/data storage are of course not that large. But investing e.g. **50 EUR for a 500 GB external USB drive** and connecting such a drive to the RPi is a simple way of using the RPi as a serious `ftp` server.

3.3 Raspberry Pi for Data Analytics

This example is about data analytics with Python (cf. http://python.org) and IPython (cf. http://ipython.org) on the RPi. Before we install it, first install the Python PIP installer by:

```
sudo apt-get install python-pip python-dev build-essential
```

3.3.1 Installing Data Analytics Libraries

Any serious data analytics effort with Python generally includes to some extent the **pandas** library (cf. http://pandas.pydata.org). To install it, **upgrade the NumPy** library first (cf. http://scipy.org):

```
sudo pip install numpy --upgrade
```

This might take quite a while (1h+) due to the library being pretty large and the RPi not being that quick in compiling it. Then **install pandas**:

```
sudo pip install pandas
```

This also takes some time (again 1h+). Also install the **matplotlib** plotting library (with some updates/dependencies) as follows:

```
sudo easy_install -U distribute
sudo apt-get install libpng-dev libjpeg8-dev libfreetype6-dev
sudo pip install matplotlib
```

And, oh wonder, this also takes quite a while to install and compile. We might want to install another useful library, namely **PyTables** (cf. http://pytables.org) for efficient I/O with Python:

```
sudo pip install numexpr
sudo pip install cython
sudo apt-get install libhdf5-serial-dev
sudo pip install tables
```

All this taken together takes a few hours in total. However, your patience will pay off: your RPi will be equipped with **state-of-the-art Python-based data analytics libraries** that can be used then for a wide range of data collection, crunching and storage tasks. Finally, install the IPython interactive analytics environment:

```
sudo pip install ipython
```

3.3.2 Interactive Data Analytics

Now **start IPython** on the shell via:

```
ipython
```

You should then see something like:

```
pi@rpi /home/ftp $ ipython
Python 2.7.3 (default, Mar 18 2014, 05:13:23)
Type "copyright", "credits" or "license" for more information.

IPython 2.3.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

Now let's retrieve some **stock quotes for the Apple stock**:

```
In [1]: import pandas.io.data as web

In [2]: aapl = web.DataReader('AAPL', data_source='yahoo')

In [3]: aapl.tail()
Out[3]:
              Open    High     Low   Close    Volume  Adj Close
Date
2014-12-18  111.87  112.65  110.66  112.65  59006200     112.65
2014-12-19  112.26  113.24  111.66  111.78  88429800     111.78
2014-12-22  112.16  113.49  111.97  112.94  45167500     112.94
2014-12-23  113.23  113.33  112.46  112.54  26028400     112.54
2014-12-24  112.58  112.71  112.01  112.01  14460200     112.01
```

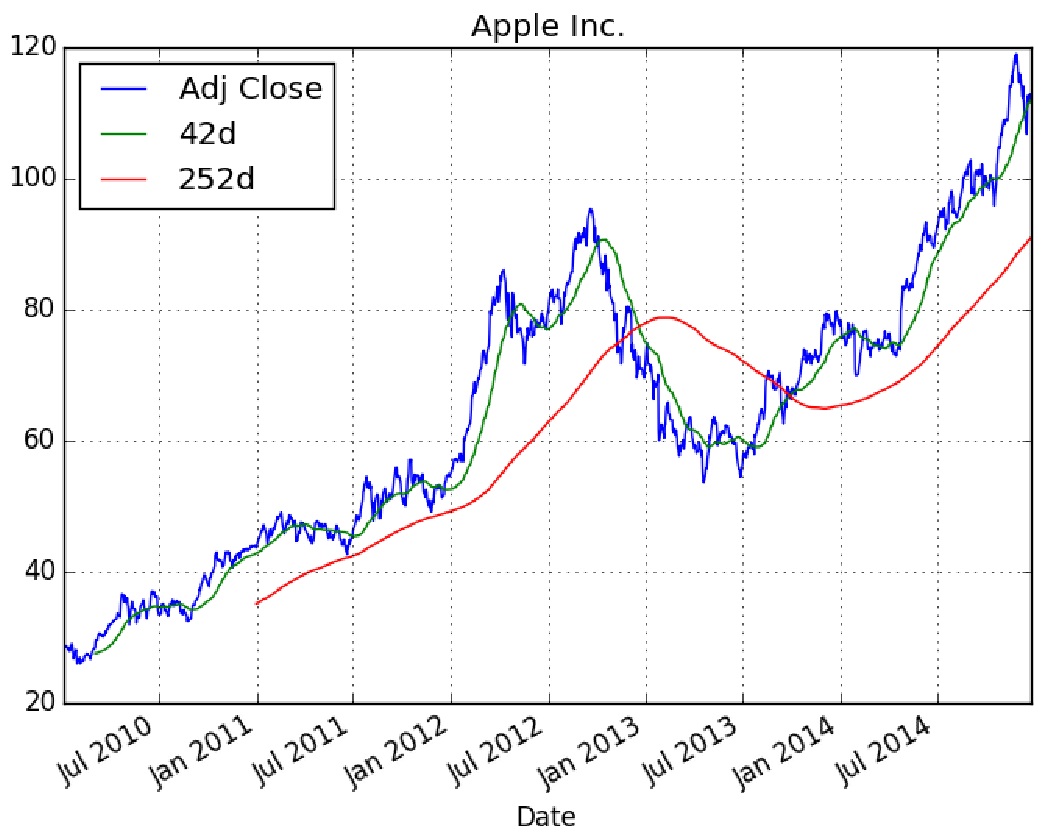
Next, let us caculate two different **moving averages** (42 days & 252 days):

```
In [4]: import pandas as pd

In [5]: aapl['42d'] = pd.rolling_mean(aapl['Adj Close'], window=42)

In [6]: aapl['252d'] = pd.rolling_mean(aapl['Adj Close'], window=252)
```

Finally, a plot of the index closing values and the moving averages:

```
In [7]: import matplotlib.pyplot as plt

In [8]: aapl[['Adj Close', '42d', '252d']].plot(title='Apple Inc.'); plt.savefig('source/aapl.png')
```

The **saved png plot** might then look like below.



Via the shell (either directly or via `ssh` access) such figures cannot be displayed. However, you could imagine to run a Web site on the RPi where the figure is included and displayed via html (see *Web Apps with Raspberry Pi*). You could also send such a graphical output/result to yourself or someone else e.g. by email or FTP transfer.

3.3.3 Fast I/O Operations

When using the RPi for data collection purposes, it might be beneficial to have efficient I/O capabilities available. This is where the PyTables library comes into play. The following is a Python script (`download link`) that collects stock data for a number of symbols and stores the data on disk in HDF5 format (cf. http://hdfgroup.org).

```
#
# Collecting and Storing Stock Price Data
# with Python/pandas/PyTables
#
# (c) Dr. Yves J. Hilpisch
# The Python Quants GmbH
#
import os
from time import time
import pandas as pd
import pandas.io.data as web

symbols = ['AAPL', 'YHOO', 'MSFT']
```

```
filename = 'data.h5'

#
# Collecting the data
#
t0 = time()
store = {}  # dictionary to store DataFrame objects

for sym in symbols:
    store[sym] = web.DataReader(sym, data_source='yahoo', start='2000/1/1')

#
# Storing data in HDF5 database
#
t1 = time()

h5 = pd.HDFStore(filename, 'w')  # open database file

for sym in symbols:
    h5[sym] = store[sym]  # write DataFrame to disk

h5.close()  # close database

t2 = time()
os.remove(filename)  # delete file on disk
#
# Output
#
print "Time needed to collect data in sec. %5.2f" % (t1 - t0)
print "Time needed to store data in sec.   %5.2f" % (t2 - t1)
```

Running the script from the shell yields an output like this:

```
pi@rpi ~ $ python data_collection.py
Time needed to collect data in sec.  1.61
Time needed to store data in sec.    1.40
```

The data gathered and stored by this Python script is not that large. The following script (`download link`) generates a set with pseudo-random sample data which is **80 MB in size** and writes it to disk.

```
#
# Storing a Larger Data Set on Disk
# with Python/pandas/PyTables
#
# (c) Dr. Yves J. Hilpisch
# The Python Quants GmbH
#
import os
from time import time
import numpy as np
import pandas as pd

filename = 'data.h5'

#
# Generating the sample data
#
t0 = time()
```

```
data = np.random.standard_normal(10000000)  # random data

df = pd.DataFrame(data)  # pandas DataFrame object

#
# Storing data in HDF5 database
#
t1 = time()

h5 = pd.HDFStore(filename, 'w')  # open database file
h5['data'] = df  # write DataFrame to disk
h5.close()  # close database file

t2 = time()
os.remove(filename)  # delete file on disk
#
# Output
#
print "Size of data set in bytes %d" % data.nbytes
print "Time needed to generate data in sec. %5.2f" % (t1 - t0)
print "Time needed to store data in sec.    %5.2f" % (t2 - t1)
```

Running this script yields an output like follows:

```
pi@rpi ~ $ python large_data_set.py
Size of data set in bytes 80000000
Time needed to generate data in sec. 10.24
Time needed to store data in sec.     9.39
```

It takes less than 10 seconds to write 80 MB of data to the SD card (times here might vary significantly depending on the card type used). You see that you can even process **larger data sets** (although not “big data”) with the RPi.

3.4 Web Apps with Raspberry Pi

Having implemented the small project about data analytics with Python, we are now well equipped to do something useful with the data analytics capabilities. In particular, we want to build a small **Web app** that serves a page where you can decide on a ticker symbol and you get back historical stock price data for that symbol.

The framework we are going to use is called **Flask** (cf. http://flask.pocoo.org/) and has become quite popular recently in the Python world. Installation in this case is straightforward:

```
sudo pip install Flask
sudo pip install flask-wtf
```

To get our Web app to be build served, we need a **Web server**. A popular choice in the Python world is **Tornado** (cf. http://www.tornadoweb.org/en/stable/). Install it via:

```
sudo pip install tornado
```

This should also be quick and straightforward. This is almost all we need to use the RPi as a Web server for Web sites or applications.

3.4.1 A First Example

Let us see if we can implement the **“Hello World!” example** of Flask and get it served with Tornado. First the Web app itself:

```
#
# From http://flask.pocoo.org
#
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"
```

As you see, a few lines of code suffice for a propor Web application—even if it is only a very small one. The `download link` for this Python module.

Next, we need to wrap the app into a **WSGI container** (cf. http://en.wikipedia.org/wiki/Web_Server_Gateway_Interface) to be served by Tornado:

```
#
# From http://flask.pocoo.org
#
from tornado.wsgi import WSGIContainer
from tornado.httpserver import HTTPServer
from tornado.ioloop import IOLoop
from flask_test import app

http_server = HTTPServer(WSGIContainer(app))
http_server.listen(5000)  # serving on port 5000
IOLoop.instance().start()
```

Again only a few lines of code. The `download link` for this script. If you execute this last script via:

```
python web_serve.py
```

you should be able to see the result when going to the **fixed IP or domain of your RPi** in combination with port 5000 (cf. *Fixed IP Address*):

```
http://xx.yy.zz.100:5000
```

or:

```
http://rpi.mydomain.net:5000
```

3.4.2 Historical Stock Price Data

This Web application retrieves historical stock price data for a user given **ticker symbol** and calculates **two different trends (moving averages)**. It then outputs a figure with the data and results as well as a HTML table with the raw data.

Let us start with the **Web app code** itself:

```
#
# Historical Stock Prices
# with the RPi using Python & Flask
#
# stock_data.py
#
# (c) Dr. Yves J. Hilpisch
# The Python Quants
#
```

```
import pandas as pd
import pandas.io.data as web
import matplotlib.pyplot as plt
from flask import Flask, request, render_template, redirect, url_for
from forms import SymbolSearch


app = Flask(__name__)

@app.route("/", methods=['GET', 'POST'])
def main():
    form = SymbolSearch(csrf_enabled=False)
    if request.method == 'POST':
        return redirect(url_for('results', symbol=request.form['symbol'],
                        trend1=request.form['trend1'],
                        trend2=request.form['trend2']))
    return render_template('selection.html', form=form)

@app.route("/symbol/<symbol>+<trend1>+<trend2>")
def results(symbol, trend1, trend2):
    data = web.DataReader(symbol, data_source='yahoo')
    data['Trend 1'] = pd.rolling_mean(data['Adj Close'], window=int(trend1))
    data['Trend 2'] = pd.rolling_mean(data['Adj Close'], window=int(trend2))
    data[['Adj Close', 'Trend 1', 'Trend 2']].plot()
    output = 'results.png'
    plt.savefig('static/' + output)
    table = data.to_html()
    return render_template('results.html', symbol=symbol,
                        output=output, table=table)


if __name__ == '__main__':
    app.run(debug=True)
```

The `download link` for this Python script.

We need a simple **WTF form** for data input (`download link`)

```
#
# Data input form
# forms.py
#

from wtforms import TextField
from wtforms.fields import SubmitField
from wtforms.validators import DataRequired
from flask.ext.wtf import Form

class SymbolSearch(Form):
    symbol = TextField('Symbol', validators=[DataRequired,])
    trend1 = TextField('Trend 1', validators=[DataRequired,])
    trend2 = TextField('Trend 2', validators=[DataRequired,])
    submit = SubmitField('Search')
```

The major **layout template** (`download link`:

```
<!-- layout.html -->
<!doctype html>
<title>Historical Stock Prices</title>
```

```
<head>

</head>

<link rel=stylesheet type=text/css
      href="{{ url_for('static', filename='style.css') }}">

<link href='http://fonts.googleapis.com/css?family=PT+Sans'
      rel='stylesheet' type='text/css'>

<link rel="shortcut icon"
      href="http://hilpisch.com/favicon.ico">

<div class=page>

  <div class=logo>
    <img src='http://hilpisch.com/tpq_logo.png' alt="TPQ Logo">
  </div>

  <div class=metanav>
    <a href="{{ url_for('main')}}">main</a>
  </div>
  {% block body %}

  {% endblock %}
</div>
```

The sub-template for the **data input** (`download link`):

```
<!-- selection.html -->
{% extends "layout.html" %}

{% macro render_field(field) %}
  <dd>{{ field.label }}
  <dd>{{ field(**kwargs)|safe }}
  {% if field.errors %}
    <ul class=errors>
    {% for error in field.errors %}
      <dd>{{ error }}</dd>
    {% endfor %}
    </ul>
  {% endif %}
  </dd>
{% endmacro %}

{% block body %}

  <form action="{{ url_for('main') }}" method=post
   enctype="multipart/form-data">
    <dl>
      <dd><h2>Search for a Ticker Symbol</h2><br>

      <dd>{{ render_field( form.symbol ) }}</dd><br>
      <dd>{{ render_field( form.trend1 ) }}</dd><br>
      <dd>{{ render_field( form.trend2 ) }}</dd><br>
      <dd>{{ render_field( form.submit ) }}</dd><br>

    </dl>
  </form>

{% endblock %}
```

And the sub-template for the **results output** (`download link`):

```
<!-- results.html -->
{% extends "layout.html" %}

{% block body %}


    <h1>Results for Symbol {{ symbol }}</h1>

    <img src="{{ url_for('static', filename='results.png') }}">

    <br><br>

    {{ table | safe }}


{% endblock %}
```

Also, a bit of **CSS** styling (`download link`)

```
/* style.css */
body { font-family: 'PT Sans', sans-serif; background: #eee; }
a, h1, h2 { color: #021A80; }
h1, h2 { font-family: 'PT Sans', sans-serif; margin: 0; text-align: center;}
h1 { font-size: 1.4em; border-bottom: 2px solid #eee; }
h2 { font-size: 1.0em; }

a:link { color: #B40404; text-decoration:none; }
a:visited { color: #B40404; text-decoration:none; }

.dataframe { margin-left: auto; margin-right: auto;}

.page { font-family: 'PT Sans', sans-serif; margin: 2em auto; width: 66em;
                  padding: 0.8em; background: white; color: #021A80}
.metanav { text-align: right; font-size: 0.8em; padding: 0.3em;
                  margin-bottom: 1em; background: #fafafa; }
.logo img { width: 30%; display: block; margin-right: auto;
            margin-left: auto; }
```

Finally, the **Tornado WSGI wrapper** for the app (`download link`):

```
#
# WSGI Wrapper for the
# Historical Stock Data App
#
# run_stock_app.py
#
from tornado.wsgi import WSGIContainer
from tornado.httpserver import HTTPServer
from tornado.ioloop import IOLoop
from stock_data import app

http_server = HTTPServer(WSGIContainer(app))
http_server.listen(8888)  # serving on port 8888
```

```
IOLoop.instance().start()
```

All these files should be placed in the following **folder structure**:

```
In [1]: import os

In [2]: for path, dirs, files in os.walk('./source/stock_app'):
   ...:     print path
   ...:     for f in files:
   ...:         print f
   ...:
./source/stock_app
forms.py
run_stock_app.py
stock_data.py
./source/stock_app/static
style.css
./source/stock_app/templates
layout.html
results.html
selection.html
```

When the working directory is `stock_app`, the following command then runs the Web app:

```
python run_stock_app.py
```

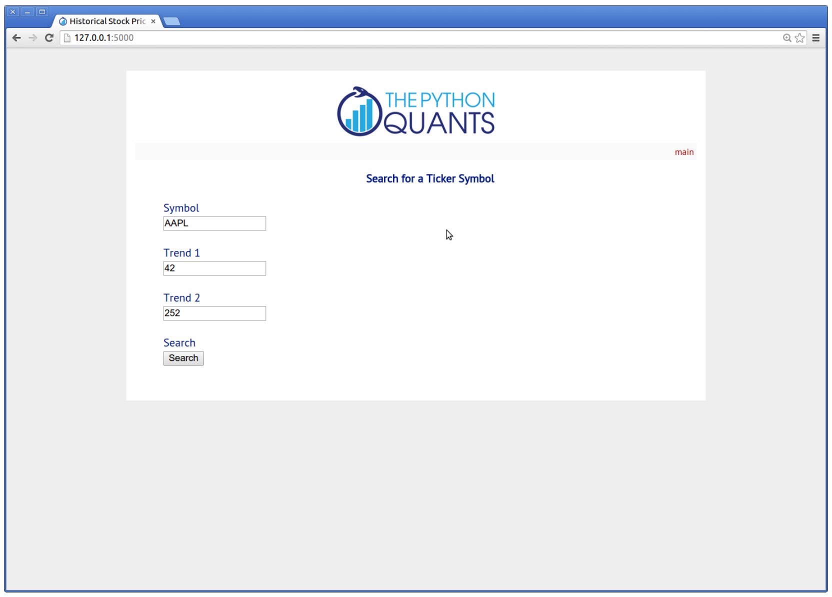
You should now be able to access the Web application via (cf. *Fixed IP Address*):

```
http://xx.yy.zz.100:8888
```

or:

```
http://rpi.mydomain.net:8888
```

The **starting/main page** for the data input might then look like (here the app is run locally):



Search for a Ticker Symbol

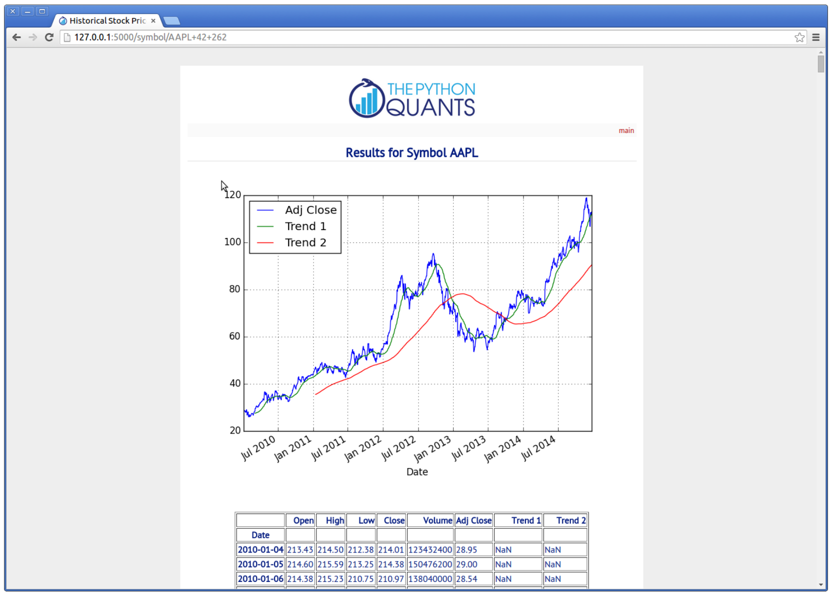
Symbol
AAPL

Trend 1
42

Trend 2
252

Search

The **results output page** looks like this:



	Open	High	Low	Close	Volume	Adj Close	Trend 1	Trend 2
Date								
2010-01-04	213.43	214.50	212.38	214.01	123432400	28.95	NaN	NaN
2010-01-05	214.60	215.59	213.25	214.38	150476200	29.00	NaN	NaN
2010-01-06	214.38	215.23	210.75	210.97	138040000	28.54	NaN	NaN

3.4.3 Generating Interactive D3 Plots

Modern Web applications generally rely on nicely rendered, interactive graphics. The following example slightly adjusts the previous one to accomplish exaclty that. The main tool used here is **plotly** (cf. http://plot.ly), a graphics engine that allows to easily transform static Python/matplotlib plots into interactive **D3.js** plots (http://d3js.org/).

You need to **install plotly** as follows:

```
sudo pip install plotly
```

You also need to create an account on the Web site http://plot.ly.

The major **changes** have be made in the main application module (`download link`):

```
#
# Historical Stock Prices
# with the RPi using Python & Flask & Plotly
#
# stock_interactive.py
#
# (c) Dr. Yves J. Hilpisch
# The Python Quants
#

import pandas as pd
import pandas.io.data as web
import matplotlib.pyplot as plt
import plotly.plotly as ply
from plotly.graph_objs import Figure, Layout, XAxis, YAxis
from flask import Flask, request, render_template, redirect, url_for
from forms import SymbolSearch

#
# Needed for plotly usage
#

ply.sign_in('yves', '65p6tn4p8i')
```

```
def df_to_plotly(df):
    '''
    Converting a pandas DataFrame to plotly compatible format.
    '''
    if df.index.__class__.__name__=="DatetimeIndex":
        x = df.index.format()
    else:
        x = df.index.values
    lines = {}
    for key in df:
        lines[key] = {}
        lines[key]['x'] = x
        lines[key]['y'] = df[key].values
        lines[key]['name'] = key
    lines_plotly = [lines[key] for key in df]
    return lines_plotly

#
# Main app
#

app = Flask(__name__)

@app.route("/", methods=['GET', 'POST'])
def main():
    form = SymbolSearch(csrf_enabled=False)
    if request.method == 'POST':
        return redirect(url_for('results', symbol=request.form['symbol'],
                                trend1=request.form['trend1'],
                                trend2=request.form['trend2']))
    return render_template('selection.html', form=form)

@app.route("/symbol/<symbol>+<trend1>+<trend2>")
def results(symbol, trend1, trend2):
    data = web.DataReader(symbol, data_source='yahoo')
    data['Trend 1'] = pd.rolling_mean(data['Adj Close'], window=int(trend1))
    data['Trend 2'] = pd.rolling_mean(data['Adj Close'], window=int(trend2))
    layout = Layout(
        xaxis=XAxis(showgrid=True, gridcolor='#bdbdbd', gridwidth=2),
        yaxis=YAxis(showgrid=True, gridcolor='#bdbdbd', gridwidth=2)
    )
    fig = Figure(data=df_to_plotly(data[['Adj Close', 'Trend 1', 'Trend 2']]),
                layout=layout)
    plot = ply.plot(fig, auto_open=False)
    table = data.to_html()
    return render_template('plotly.html', symbol=symbol,
                            plot=plot, table=table)


if __name__ == '__main__':
    app.run(debug=True)
```

We also need to **adjust the results output template file** (`download link`):

```
<!-- plotly.html -->
{% extends "layout.html" %}

{% block body %}
```

```
    <h1>Results for Symbol {{ symbol }}</h1>

    <iframe

    width=100% height="650" frameborder="0" seamless="seamless" scrolling="no"

    src="{{ plot }}.embed?width=100%&height=650">

    </iframe>

    <br><br>

    {{ table | safe }}


{% endblock %}
```

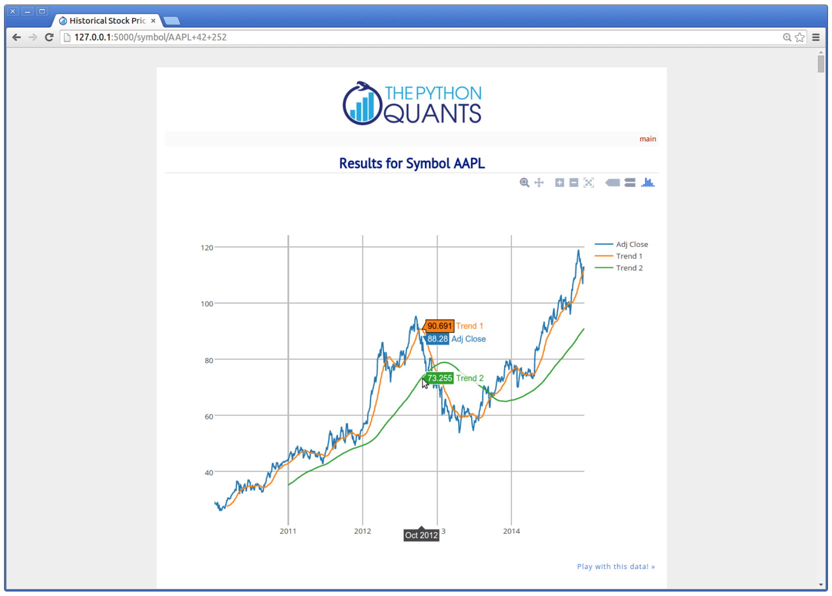
Also, the **WSGI wrapping** is to be adjusted slightly (`download link`):

```
#
# WSGI Wrapper for the
# Historical Stock Data App (Interactive)
#
# run_stock_int.py
#
from tornado.wsgi import WSGIContainer
from tornado.httpserver import HTTPServer
from tornado.ioloop import IOLoop
from stock_interactive import app

http_server = HTTPServer(WSGIContainer(app))
http_server.listen(8888)  # serving on port 8888
IOLoop.instance().start()
```

Everything else remains the same. Your **folder structure** should now look like follows:

```
In [3]: for path, dirs, files in os.walk('./source/stock_int'):
   ...:     print path
   ...:     for f in files:
   ...:         print f
   ...:
./source/stock_int
forms.py
run_stock_int.py
stock_interactive.py
./source/stock_int/static
style.css
./source/stock_int/templates
layout.html
plotly.html
selection.html
```

If everything runs as desired, the **results page** of the interactive version should look like below (here the app runs locally):


Historical Stock Pric
127.0.0.1:5000/symbol/AAPL+42+252
THE PYTHON QUANTS
main
Results for Symbol AAPL
Adj Close
Trend 1
Trend 2
90.691 Trend 1
88.28 Adj Close
73.255 Trend 2
120
100
80
60
40
2011
2012
Oct 2012
2014
Play with this data! »

CHAPTER

FOUR

ABOUT THE AUTHOR

Yves Hilpisch is managing partner of The Python Quants GmbH (Germany) and co-founder of The Python Quants LLC (New York City). The Python Quants provide, among others, the **Python Quant Platform** as a solution for browser-based, interactive, collaborative financial analytics (cf. http://quant-platform.com). On this platform (for which free trials are available) you can also immediately try our open source financial analytics library DX Analytics (http://dx-analytics.com).

CHAPTER

FIVE

COPYRIGHT & DISCLAIMER