A Guide to Installing OpenMDAO on an RHEL (or CentOS) Machine with No Admin Rights

By Keith G. Marsteller (Keith.G.Marsteller@nasa.gov)

Before you begin the installation guide, if you have administrator rights over your machine, don't follow this guide, just go make sure your machine has python2.7 and the OpenMDAO pre-requisites(http://openmdao.org/docs/getting-started/requirements.html), and go get OpenMDAO from http://openmdao.org/downloads/recent.

This guide is for those poor souls who are stuck with an old RHEL(or CentOS) installation, no admin rights, and python older than 2.7 as their default system python. (OpenMDAO will not work with anything other than python 2.7.) Here's how you may circumvent the restrictions of such a system.

OVERVIEW:

The basic idea of how we get around the obstacles of no admin rights, and the wrong python is that we will use your user account to house the creation of an independent-from-the-system python install, we will alter our path environment variable to favor this installation over the system installation, and then install pip to help us install numpy, scipy and matplotlib. Once all that has been done, we will acquire and install OpenMDAO, and make sure that its installation is successful.

So, let's begin!

BASH

Before you do anything else, make sure you're running in the bash shell, otherwise, nothing that follows will work! I mention this first, because by default on these systems, you'll be running in tcsh:

```
[kmarstel@RHEL ~]$ echo $SHELL
/bin/tcsh

[kmarstel@RHEL ~]$ bash
bash$
```

INSTALL PYTHON 2.7.8

In the following steps, we are going to download the source code for Python, unpackage the source code, compile the source code, and install the source code.

Download:
Make sure that you are in your */home/<username>* directory, or whatever directory you want to put this python installation in. Just know that if you do, from here on out in the document, you'll need to replace */home/<username>* with whatever directory you're using.

Download the compressed source code package using `wget`(in this case, python 2.7.8):

```
wget http://www.python.org/ftp/python/2.7.8/Python-2.7.8.tar.xz --no-check-certificate
```

Unpackage:
Extract the files from this source code package:

```
xz -d Python-2.7.8.tar.xz
tar –xvf Python-2.7.8.tar
```

Once extracted, a new directory of source code, Python-2.7.8 will be there. To start compiling and installing, change directories into it.

```
cd Python-2.7.8
```

Compile & Install
What we have done is download and unpackage the source code for Python. Now we are going to compile it, and install it in our home directory.

Configure and build the application using autoconf (*configure*) and *make*. By default, files would be installed in */usr/local*. We use the *–prefix* argument to *configure* to change that to be a new directory */home/<username>/python2.7*. To be clear: this is a different location that the Python-2.7.8 source code directory that you have in your home directory right now, and in fact this directory doesn't exist yet. You could create this directory in `/home/<username>` by typing "`mkdir /home/<username>/python2.7`", but if you don't create it, the install process will create it for you. In any case, your next command is:

```
./configure --prefix=/home/<username>/python2.7
```

Next, build (compile) the source. this procedure will take several minutes:

```
make
```

After building everything with the first make, we make sure that it gets moved to our alternate install venue in */home/<username>/python2.7*:

```
make altinstall
```

It is at this point, once python installation has finished, that you should have a */home/username/python2.7/bin* directory in existence. Within that is an executable called python2.7. Now you need to ensure that this python is found before your system python when you execute the command "`python2.7`".

SETTING UP SYSTEM PATH

Once you have a local version of python installed, you're going to be editing (if it already exists) or creating (if it doesn't exist) a file in your home directory called `.bashrc`. (**NOTE: on CentOS, make or edit *.bash_profile*)** This is a file that is run when you enter the bash shell. I use my .bashrc to set up my path to use my local python every time I login. My .bashrc/.bash_profile file looks like this.

```
# User specific aliases and functions
export PATH=/home/kmarstel/python2.7/bin:$PATH
```

What this one line does is prepend the home directory python to the front of the PATH. That way, when you type "*python2.7*" and it looks for a python2.7 to execute, it finds the one in your */home/<username>/python2.7/bin* area first, instead of finding */usr/bin/python* which helps us circumvent having to use the system-level python.

Once you do edit or create the .bashrc file, however, you will need to either source the file ("*source .bashrc*") or start a new terminal and run bash again to get the contents working. To be sure the .bashrc is working at changing the path, when you enter the bash shell, you can simply echo the path variable, and read it to make sure that your */home/<username>/python2.7/bin* directory appears first.

```
bash$ echo $PATH

/home/kmarstel/python2.7/bin:/usr/lib64/qt-
3.3/bin:/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin
:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/eng
apps/bin
```

MAKING SURE OUR PYTHON INSTALL is WORKING

OK, so we have installed python, and tried to make sure we are finding our installed python first whenever we invoke the python command. Let's find out if we succeeded by trying the *which* command. The *which* command is helpful to find

out which of possibly many installs of an executable is the one that would be invoked if that command were typed at this location with this $PATH at this time.

```
bash$  which python2.7
/home/kmarstel/python2.7/bin/python2.7
```

This should report back the python that you just installed in your /home/<username> directory, as shown above. In most cases, it should be called "`python2.7`". If it doesn't report back as the right python or python2.7, check your $PATH again, and/or manually inspect that your python is in the place you think it is. All systems won't be the same, path-wise, so keep in mind that you may need to customize your path to fit where your python is installed in your situation rather than following this guide to the letter.

Double-check
Let's check on which version of the gcc compiler we have:

```
bash$ gcc –v
gcc version 4.9.1 (GCC)
```

If we run our python2.7, we should see that it was built today with this compiler and this version (just another clue that we did things correctly.):

```
bash$ python2.7
Python 2.7.8 (default, Jan 12 2015, 10:20:25)
[GCC 4.9.1] on linux2
bash$ quit()
```

Cleanup:
Assuming that your install went well, you can now remove the source code directory (Python-2.7.8) and any leftover zip/tar files. The `/home/<username>/python2.7` directory will be all that you need.

INSTALL PIP

Assuming that we are in good shape with the newly-installed python, it is time to get pip. Pip is an installer that we use to install OpenMDAO prerequisites into our python environment. Start off in your `/home/<username>` directory. Download the installation file using wget:

```
wget --no-check-certificate http://bootstrap.pypa.io/get-pip.py
```

Install pip using the python we've just installed:

```
python2.7 get-pip.py
```

NOTE: pip installs things into the python installation of the python that installs pip. Therefore, it is critical that the "python2.7" command used to run get-pip.py is using the python2.7 that lives in your home directory, because the python that is used to install pip will be the destination for all subsequent pip installation commands. If `which python2.7` is wrong, the prerequisites we are about to install will be installed to an unintended python environment. This will create confusion and frustration.

INSTALL NUMPY/SCIPY/MATPLOTLIB

The work that we have done so far should make this section relatively painless, in that we can just type commands, and let them take care of the installations themselves. Again, we will be installing numpy/scipy/matplotlib into the python that we just made, and we can check that at the end.

First, let's save some hassle by making sure we have the right pip:

```
bash$ which pip
/home/kmarstel/python2.7/bin/pip
```

It obviously needs to be the pip in our locally-installed python in order for this to work. If that's correct, then the rest should be easy. However, WARNING, each of these commands will take several minutes to complete:

```
pip install numpy
pip install scipy
pip install matplotlib
```

Let's test to make sure those things were installed to the right place. A double-check will be to run python, import numpy (no response means success), and then check a property of numpy, __file__ (that's two underscores on each side of the word 'file'). If pip installed things correctly, "`numpy.__file__`" should show itself to be installed in your python install's site-packages :

```
bash$ python2.7

Python 2.7.8 (default, Jan 12 2015, 10:20:25)
[GCC 4.9.1] on linux2

>>> import numpy
>>> numpy.__file__
```

```
'/home/kmarstel/python2.7/lib/python2.7/site-
packages/numpy/__init__.pyc'
```

With those things installed, we're ready to get OpenMDAO.

GETTING OPENMDAO

Getting OpenMDAO is easy, all that is needed is to download a go-openmdao-<version>.py file and then run python2.7 on it. Get the latest release at http://openmdao.org/downloads/recent, or you can `wget` it using an address like:

wget http://openmdao.org/releases/<release_number>/go-openmdao-<release_number>.py

or this real example:

wget http://openmdao.org/releases/0.10.3.2/go-openmdao-0.10.3.2.py

Then, the installation is easy:

python2.7 go-openmdao-0.10.3.2.py

This will start the installation process, which takes a few minutes, and a LOT of text will scroll by, and that text may contain many warnings. Don't let those warnings scare you. At the end of the process, read the messages at the very end to make sure that all the packages installed correctly.

If things went wrong, there will be explicit messages at the very end of the build output (you don't need to search through it) saying,

```
the following packages failed to install:
 [openmdao.package1, openmdao.package2]
```

If things worked, you'll see no errors and a message like this:

```
 The OpenMDAO virtual environment has been installed in
 /home/<username>/openmdao-0.10.3.2

From /home/<username>/openmdao-0.10.3.2, type:
. bin/activate
```

So, just follow the instructions as printed there for activation:

```
bash$ cd openmdao-0.10.3.2
bash$ . bin/activate
```

Activation will change your bash prompt, to remind you that you're now working within the openmdao virtual environment.

```
(openmdao-0.10.3.2)-bash$
```

Then, you can start using OpenMDAO. To do a quick (5 minute) test of OpenMDAO, type, at the activated prompt:

```
(openmdao-0.10.3.2)-bash$ openmdao test -v
```

Passing tests indicate that all is well with your install. If you have problems with your installation, please feel free to ask a question at http://openmdao.org/forum, or send an email to support@openmdao.org

To get started working with OpenMDAO, check out our documentation by typing `openmdao docs` at an activated command prompt (will launch a browser if possible in which to view them), or by visiting our website http://openmdao.org/docs.

NOTES

1. Some RHEL machines are shared by more than one user. To save the trouble of each user having to follow the above procedures, one user could complete the installation, and then modify the permissions on his/her home directory to allow others to see the local python installation. Then all the other users can simply change their paths to prepend your home directory's python.

2. You might get clever and decide that instead of changing your path in the .bashrc, that you're just going to create an alias for the "python" command to call the python in your home directory. Well, that works great for the one command that you'll use to build OpenMDAO (above). But once you activate OpenMDAO's virtual environment, you want OpenMDAO to use the python that is built down inside there…and with an alias, you will instead be pointing back up out of your OpenMDAO virtualenv to the python in your home directory, and nothing will work properly.

3. When you run the openmdao test –small on RedHat, there will be one test that we expect to fail, due to a difference in outputs between our supported distribution, which is Ubuntu, and RedHat:

```
======================================================================
FAIL: test_badcmd (openmdao.lib.components.test.test_extcode.TestCase)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/kmarstel/openmdao-0.9.7/lib/python2.7/site-
packages/openmdao.lib-0.9.7-
```

```
py2.7.egg/openmdao/lib/components/test/test_extcode.py", line 331, in
test_badcmd
    self.assertEqual(str(exc), msg)
AssertionError: '[Errno 13] Permission denied' != '[Errno 2] No such file or
directory'
```

4. If you are having trouble installing things, you might need to ask your sysadmin to verify the installation of the following packages to the system level:

sudo yum groupinstall "Development tools"
sudo yum install zlib-devel bzip2-devel openssl-devel sqlite-devel
sudo yum install readline-devel tk-devel gdbm-devel db4-devel libpcap-devel
sudo yum install xz-devel blas-devel lapack-devel