Debian packages of ROOT.

BV

[2014-10-20 Mon 09:20]

1 ROOT Debian packages compared to others

Christian Christensen has done a lot of great work in developing Debian packages of ROOT. They are distributed as part of both Debian proper and Ubuntu. Since they are proper Debian packages they present a ROOT environment which differs a bit from what one has when installing from source. In particular Debian policy requires that no special environment variables must be set to use a package where "standard" ROOT installs typically require `ROOTSYS` to be set to the base installation location and various `*PATH` variables to be added to.

Life goes on and support for these packages has waned somewhat in the community so things are not always so smooth. This has even led to Debian dropping the ROOT packages from time to time. Below collects some information on how to get over the bumps in using these packages

2 Building Debian packages from source

t.b.d.

3 Using Debian ROOT packages

3.1 PyROOT, ROOT Python Bindings

In principle, one need not do anything special to use PyROOT:

```
$ python

>>> import ROOT
>>> ROOT.TBrowser()
```

3.1.1 No module named ROOT

At least in Ubuntu 14.04 naive importing of the `ROOT` module fails because the `ROOT.py` file is not found.

```
$ python -c 'import ROOT'
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ImportError: No module named ROOT
```

Here is it's location:

```
$ dpkg -L libroot-bindings-python5.34 | grep ROOT.py
/usr/lib/x86_64-linux-gnu/root5.34/ROOT.py
/usr/share/python-support/root/ROOT.py
```

The two copies are identical.

According to Debian policy `python-support` is deprecated. It references the doc area of the package of the same name which also says this location is deprecated but supported. Neither reverence what is now in favor.

After a few random searches and poking around, it seems like the place these sort of things are supposed to live is now under `dist-packages`. Symlinking the `ROOT` module there seems to let things work

```
$ sudo ln -s /usr/share/python-support/root/ROOT.py /usr/lib/python2.7/dist-packages/
$ python -c 'import ROOT'
$ echo $?
0
```

3.2 PyROOT and `ipython`

The `ipython` program gives you a fantastic Python command line (and other goodies). Use it.

```
$ sudo apt-get install ipython
$ ipython
...
In [1]: import ROOT

In [2]: ROOT.TBro<TAB>
```

```
In [2]: ROOT.TBrowser()
Out[2]: <ROOT.TBrowser object ("Browser") at 0x35b8a70>

  In [3]: ROOT.TBro<TAB>
  In [3]: ROOT.TBrowser
```

Tab completion works but take note that many of the top-level objects in the `ROOT` module are autoloaded and so will not be around until the are first explicitly referenced. Online documentation is exposed with ipython's usual "?". As the example below shows there is not always docstrings but a lot can be learned from the (C++) calling prototypes.

```
In [3]: ROOT.TBrowser?
Type:       TBrowser_meta
String Form:<class 'ROOT.TBrowser'>
File:       /usr/lib/python2.7/dist-packages/ROOT.py
Docstring:  <no docstring>
Constructor information:
 Definition:ROOT.TBrowser(self, const char* name, void* obj, TClass* cl, const char* o
 Docstring:
    TBrowser::TBrowser(const char* name = "Browser", const char* title = "ROOT Object
    TBrowser::TBrowser(const char* name, const char* title, UInt_t width, UInt_t heigh
    TBrowser::TBrowser(const char* name, const char* title, Int_t x, Int_t y, UInt_t w
    TBrowser::TBrowser(const char* name, TObject* obj, const char* title = "ROOT Objec
    TBrowser::TBrowser(const char* name, TObject* obj, const char* title, UInt_t width
    TBrowser::TBrowser(const char* name, TObject* obj, const char* title, Int_t x, Int
    TBrowser::TBrowser(const char* name, void* obj, TClass* cl, const char* objname =
    TBrowser::TBrowser(const char* name, void* obj, TClass* cl, const char* objname, c
    TBrowser::TBrowser(const char* name, void* obj, TClass* cl, const char* objname, c
```

3.3 PyROOT and `virtualenv`

The command `virtualenv` produces a directory that is configured to be a mini-environment for Python programs. It is a great way to try out Python packages w/out disturbing the system or to allow multiple, independent installations to coexist in your user account. By default `virtualenv` does not expose any system packages to the environment it makes. This is really good for developing a Python package as it allows the developer to understand what dependencies their package may need. But, it does block the user from accessing PyROOT from ROOT installed as a system package.

To create a virtual environment with access to system Python packages one needs to pass a flag:

```
$ virtualenv --system-site-packages venvdir
```

If you have already created a `venvdir` and did not instruct the use of system site packages you can re-run the command with this flag added. Then setup and use/test it in the normal manner:

```
$ source venvdir/bin/activate
$ python -c 'import ROOT'
```