GTAC Tracker Documentation

Release 1

Brian Koebbe

June 02, 2015

1 **STS Hardware** **3**

2 **STS Development** **5**
2.1 Quick Start 5
2.2 Setting up your virtual environment 6

3 **Pyhton access to the STS** **7**
3.1 Quick and Dirty Hard Coding 7
3.2 More Portable using the ‘workon’ method 7
3.3 Manuevering Around the Models 8

4 **Source Code** **9**

5 **Indices and tables** **11**

NextGen Sequencing Tracking System (the “STS”) provides a web-based, database-backed system for managing the NextGen sequencing operations of the GTAC. Each step of the sequencing process, from sample submission to publishing of the final analysis results, are thoroughly documented in a database, and made available for reporting, billing, and process analysis.

test

Contents:

CHAPTER 1

STS Hardware

The GTAC STS is currently hosted on a CGS Dell PowerEdge M600 with 16GB RAM.

The software itself is stored on 1 Unit (4.6TB, RAID 10) of CGS Storage and a full backup is performed daily.

The server is accessible to members of GTAC via the CGS Cluster using ssh:

```
ssh gtac.wustl.edu
```

CHAPTER 2

STS Development

The GTAC STS was created using multple open-source software tools, languages, and practices. To extend, enhance and gereally make the most of the STS, a general knowledge of the following components is necessary.

Python Programming Language (v2.7) Nearly all code used in the STS is python. The rather large official python tutorial is a good place to start. Knocking on my door works well too.

Django Web Framework (v1.3) Django is the foundation of the STS. Once you've set up your environment (see *Setting up your virtual environment*) I recommend closely following the Django Tutorial so you have an understanding of the basic Django concepts.

PostgreSQL (v8.4) and/or SQLite (v3) The production STS stores its data in PostgreSQL, but your development instance will most likely use SQLite. A basic intro to SQL may be all that you need.

Mercurial All development changes are captured using the Mercurial (hg) distributed source control management tool. The quick start and cheatsheets are probably all you need to get going.

2.1 Quick Start

1. Make sure python-virtualenv is installed
2. Build and activate your environment:

```
$ mkdir ~/prj
$ cd ~/prj
~/prj$ mkvirtualenv sts
~/prj$ workon sts
(sts)~/prj$
```

3. Clone your own GTAC STS source code repository:

```
(sts)~/prj$ hg clone http://bitbucket.org/koebbe/sts sts-wc
(sts)~/prj$ cd sts-wc
(sts)~/prj/sts-wc$ hg clone http://bitbucket.org/koebbe/djmenu menus
(sts)~/prj/sts-wc$ hg clone http://bitbucket.org/koebbe/sts-simpla simpla  *PRIVATE*
```

4. Install your python and django packages:

```
(sts)~/prj/sts-wc$ pip install -r requirements.txt
```

Note: SQLite3 developement libraries will need to be installed to build pysqlite

5. Build your development database:

```
    (sts)~/prj/sts-wc$ ./manage.py syncdb
    (sts)~/prj/sts-wc$ ./manage.py migrate
```

6. Start up your GTAC STS website:

```
    (sts)~/prj/sts-wc$ ./manage.py runserver
```

7. Browse to http://localhost:8000/
8. ...
9. **Profit!**

2.2 Setting up your virtual environment

When working on extending/enhancing the STS, having a consistent and dependable STS development environment is critical. The following is a quick explanation of how to set one up.

2.2.1 virtualenv

The following is an explanation of virtualenv from its website...

`virtualenv` is a tool to create isolated Python environments.

The basic problem being addressed is one of dependencies and versions, and indirectly permissions. Imagine you have an application that needs version 1 of LibFoo, but another application requires version 2. How can you use both these applications? If you install everything into `/usr/lib/python2.7/site-packages` (or whatever your platform's standard location is), it's easy to end up in a situation where you unintentionally upgrade an application that shouldn't be upgraded.

Or more generally, what if you want to install an application *and leave it be*? If an application works, any change in its libraries or the versions of those libraries can break the application.

Also, what if you can't install packages into the global `site-packages` directory? For instance, on a shared host.

In all these cases, `virtualenv` can help you. It creates an environment that has its own installation directories, that doesn't share libraries with other virtualenv environments (and optionally doesn't access the globally installed libraries either).

The basic usage is:

```
$ virtualenv ENV
```

This creates `ENV/lib/pythonX.X/site-packages`, where any libraries you install will go. It also creates `ENV/bin/python`, which is a Python interpreter that uses this environment. Anytime you use that interpreter (including when a script has `#!/path/to/ENV/bin/python` in it) the libraries in that environment will be used.

It also installs 'Setuptools <http://peak.telecommunity.com/DevCenter/setuptools>'_into the environment.

A new virtualenv also includes the pip installer, so you can use *ENV/bin/pip*' to install additional packages into the environment.

CHAPTER 3

Pyhton access to the STS

Python sccess to STS python models is now available from the CGS Cluster!

An STS python virtual environment has been prepared on the cluster which will facilitate access to the STS by scripts running on sandboxes or jobs running on the cluster. This virtualenv includes all the python modules necessary to use the Django API to get to the STS data.

The virtual environment is located in: /srv/gtac/sts/envs/sts-cluster

3.1 Quick and Dirty Hard Coding

The easiest way to use this environemnt is by using the following for the top of your python scripts... let's call it 'example.py':

```
#!/srv/gtac/sts/envs/sts-cluster/bin/python

import os
import sys
os.environ['DJANGO_SETTINGS_MODULE']='settings'
sys.path.append('/srv/gtac/sts/sts')
```

3.2 More Portable using the 'workon' method

To make the script more portable and future-proof you could instead load up the environment before running the script:

```
$ export WORKON_HOME=/srv/gtac/sts/envs
$ workon sts-cluster
```

Then your python programs only need the usual line:

```
#!/usr/bin/env python
```

Using this method you can also get to the STS from the python CLI. For example, to grab the latest submission:

```
$ python
Python 2.7.2+ (default, Oct  4 2011, 20:06:09)
[GCC 4.6.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from sample import models
>>> s = models.Submission.objects.order_by('-submitted')[0]
```

3.3 Manuevering Around the Models

You can use http://gtac.wustl.edu/sts/admin/doc/models to help you manuever around the data. For example, let's try and make a report showing the number of samples each client has submitted.

1. Grab the Clients

   http://gtac.wustl.edu/sts/admin/doc/models shows that the 'client' model is in 'base'. So:

```
from base import models as basemodels

clients = basemodels.Client.objects.all()
```

2. Follow the trail from Client to Sample

   Browsing around http://gtac.wustl.edu/sts/admin/doc/models/sample.submission we see that there are 1 or more 'samples' ('sample_set.all()') per 'submission' and that there is one client per submission.

   From this page we can follow the trail to 'client' by clicking on 'base.Client'. From here we see that 'client' has access to 'submissions' via 'submission_set.all()'

   Now we can grab a client's submissions:

```
for client in clients:
    clients_submissions = client.submission_set.all()
```

Source Code

Checkout a copy of the STS:

```
hg clone http://gtac.wustl.edu/hg/gtac-lims
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search