SatNOGS Documentation

Release 1

SatNOGS

May 07, 2017

Contents

This is the documentation for all parts of the SatNOGS project.

Contents:

CHAPTER 1

satnogs-client

Contents:

SatNOGS client architecture

Overview

SatNOGS client is the part of our software stack that:

- Fetches observation jobs from the network.
- Schedules locally when the observation starts/ends.
- Does orbital calculation for the position of the observer and the tracked object (using `PyEphem`).
- Sends `rotctl/rigctl` commands to control **SatNOGS** rotator.
- Spawns processes to run demodulation/decoding software with the signal received as input.
- Posts observation data back to the network.

Modules

Following the paradigm of **SatNOGS** being extensively modular, **SatNOGS** client is designed to have discrete modules with specific functionality.

`scheduler`

- Build using `apscheduler` library.
- Stores tasks in `sqlite`.

Tasks

- `get_jobs`: Queries **SatNOGS Network API** to get jobs scheduled for the ground station.
- `spawn_observation`: Initiates an `Observer` instance and runs the observation.
- `post_observation_data`: Gathers output files, parses filename and posts data back to **SatNOGS Network API**.

observer.observer

Given initial description of the observation (`tle`, `start`, `end`)

- Checks input for sanity.
- Initializes `WorkerTrack` and `WorkerFreq` instances that start `rigctl/rotctl`. communication using `trackstart` method.
- Starts/Stops GNU Radio script (`gr-satnogs`), which collects the data.
- Processes produced data from observation (ogg file, waterfall plotting).

observer.worker

- Facilitates as a worker for `rigctl/rotctl`.
- Is the lowest abstraction level on `rigctl/rotctl` communications.
- Tracks object until end of observation is reached.

observer.orbital

- Implements orbital calculations using `PyEphem`.
- Provides `pinpoint` method the returns alt/az position of tracked object.

bin/scripts

- `satnogs-client`: Run the scheduler queue in the background and fetch jobs from the network.

satnogsclient Package

satnogsclient Package

settings Module

Subpackages

observer Package

commsocket Module

class satnogsclient.observer.commsocket.**Commsocket**(*ip*, *port*)
Handles connectivity with remote ctl demons Namely: rotctl and rigctl

accept()

bind()

buffer_size

connect()

disconnect()

ip

is_connected

listen()

port

receive(*size*)

send(*message*)

send_not_recv(*message*)

tasks_buffer_size

observer Module

class satnogsclient.observer.observer.**Observer**

frequency

location

observation_end

observation_id

observation_ogg_file

observation_raw_file

observation_waterfall_file

observation_waterfall_png

observe()
Starts threads for rotcrl and rigctl.

plot_waterfall()

poll_gnu_proc_status()

remove_waterfall_file()

rename_ogg_file()

rig_ip

rig_port

rot_ip

rot_port

run_rig()

run_rot()

setup(*observation_id*, *tle*, *observation_end*, *frequency*, *user_args*, *script_name*)
Sets up required internal variables. * returns True if setup is ok * returns False if issue is encountered

tle

orbital Module

satnogsclient.observer.orbital.**pinpoint**(*observer_dict*, *satellite_dict*, *timestamp=None*)
Provides azimuth and altitude of tracked object.

args: observer_dict: dictionary with details of observation point. satellite_dict: dictionary with details of satellite. time: timestamp we want to use for pinpointing the observed object.

returns: Dictionary containing azimuth, altitude and "ok" for error detection.

worker Module

class satnogsclient.observer.worker.**Worker**(*ip*, *port*, *time_to_stop=None*, *frequency=None*, *proc=None*)
Class to facilitate as a worker for rotctl/rigctl.

SLEEP_TIME = 0.1

check_observation_end_reached()

is_alive
Returns if tracking loop is alive or not.

observer_dict = {}

satellite_dict = {}

send_to_socket()

trackobject(*observer_dict*, *satellite_dict*)
Sets tracking object. Can also be called while tracking to manipulate observation.

trackstart(*port*, *start_thread*)
Starts the thread that communicates tracking info to remote socket. Stops by calling trackstop()

trackstop()
Sets object flag to false and stops the tracking thread.

class satnogsclient.observer.worker.**WorkerFreq**(*ip*, *port*, *time_to_stop=None*, *frequency=None*, *proc=None*)
Bases: *satnogsclient.observer.worker.Worker*

send_to_socket(*p*, *sock*)

class satnogsclient.observer.worker.**WorkerTrack**(*ip*, *port*, *time_to_stop=None*, *frequency=None*, *proc=None*)
Bases: *satnogsclient.observer.worker.Worker*

send_to_socket(*p*, *sock*)

scheduler Package

scheduler Package

tasks Module

satnogsclient.scheduler.tasks.**add_observation**(*obj*)

satnogsclient.scheduler.tasks.**ecss_feeder**(*port*)

satnogsclient.scheduler.tasks.**exec_rigctld**()

satnogsclient.scheduler.tasks.**get_jobs**()

satnogsclient.scheduler.tasks.**get_observation**(*id*)

satnogsclient.scheduler.tasks.**get_observation_list**()

satnogsclient.scheduler.tasks.**kill_cmd_ctrl_proc**()

satnogsclient.scheduler.tasks.**kill_netw_proc**()

satnogsclient.scheduler.tasks.**kill_wod_thread**()

satnogsclient.scheduler.tasks.**post_data**()

satnogsclient.scheduler.tasks.**rigctld_subprocess**()

satnogsclient.scheduler.tasks.**signal_term_handler**(*a*, *b*)

satnogsclient.scheduler.tasks.**spawn_observer**(**args*, ***kwargs*)

satnogsclient.scheduler.tasks.**start_wod_thread**()

satnogsclient.scheduler.tasks.**status_listener**()

satnogsclient.scheduler.tasks.**success_message_to_frontend**()

satnogsclient.scheduler.tasks.**task_feeder**(*port*)

satnogsclient.scheduler.tasks.**wod_listener**()

SatNOGS Client Installation

Installing SatNOGS on a Raspberry Pi 3

This tutorial assumes the following:

1. You have a Raspberry Pi 3 with external power (5V, 2A).
2. USB keyboard, HDMI screen, HDMI cable, network cable (Wi-Fi isn't working on Fedora for now) and a Class 10 SDHC card at least 8GB.
3. One of the following sdr devices: RTL-SDR or USRP B200.
4. You have an account and a ground station registered on either network.satnogs.org or network-dev.satnogs.org. This is needed for getting your ground station ID number and your SatNOGS Network API key.

1. Prepare Raspberry Pi

Step 1.1: Download fedora minimal or server RPi image (current 25) from ARM Fedora Project (server edition provides a nice web interface, admin console, with several stats and SSH access).

Step 1.2: Connect SD card to your computer/laptop and prepare it as described at Fedora Wiki (don't forget to resize the root partition).

Step 1.3: Attach sdcard to your RPi, plug in the HDMI cable, keyboard and ethernet and turn on the HDMI screen. Plug RPi to the power source.

Step 1.4: Fedora installation starts, follow the steps that show up in the screen. You'll have to setup:

- root password
- network connection
- timezone and ntp server, add at least one, *pool.ntp.org* is suggested
- a new user, e.g. *satnogs*. Don't forget to set administrator flag and add user to *dialout* group (needed for having access to sdr device).

Step 1.5: From now on you are able to access you RPi directly or through SSH. You can also use admin console if you have selected the fedora server version.

Step 1.6: Update fedora package to the latest version by running:

```
sudo dnf -y update
```

Step 1.7: Install dependencies for gr-satnogs and satnogs-client:

```
sudo dnf install -y util-linux-user git gcc redhat-rpm-config python-devel redis␣
↪vorbis-tools hamlib gnuradio gnuradio-devel cmake swig fftw3-devel gcc-c++ cppunit␣
↪cppunit-devel doxygen gr-osmosdr libnova libnova-devel gnuplot libvorbis-devel␣
↪libffi-devel openssl-devel libpng-devel
```

Step 1.8: In order to expand the lifetime of the SD Card, edit /etc/fstab file with your favourite editor:

- Comment out the line of the swap partition
- Only if you used the "minimal" Fedora installation (not the "server" build), change options of root partition line (/ ext4) from *defaults,noatime* to *defaults,noatime,commit=1800*. This change means that changes on root partition will be written on SD Card every 30min
- Move /var/log and /var/tmp directories to memory by adding the following two lines:

```
tmpfs /var/tmp tmpfs defaults,noatime,nosuid,size=20m 0 0
tmpfs /var/log tmpfs defaults,noatime,nosuid,mode=0755,size=80m 0 0
```

Step 1.9: Automate creating of redis directory in /var/log path after boot by running:

```
sudo sh -c 'echo "#Type Path                Mode UID   GID   Age Argument" > /etc/
↪tmpfiles.d/logdirs.conf'
sudo sh -c 'echo "d     /var/log/redis      0750 redis redis 1d  -" >> /etc/tmpfiles.
↪d/logdirs.conf'
```

Step 1.10: Enable and start Redis service in order to run automatically on startup:

```
sudo systemctl enable redis.service
sudo systemctl start redis.service
```

Step 1.11: Configure automatic cleanup of old data (while this is an optional step, if old files are not cleaned out regularly you ru

- As-is this will clean out files older than 1 week. Adjust mtime to your liking
- Create /etc/cron.daily/satnogs with your favorite editor and add the following:

```
#!/bin/sh
find /tmp/.satnogs/data -type f -mtime +7 -delete
```

- Then run:

```
sudo chmod +x /etc/cron.daily/satnogs
```

2. Install gr-satnogs

SatNOGS Client uses GNU Radio scripts in order to get observation data from satelites, gr-satnogs provide this functionality.

Step 2.1:: Install gr-satnogs by running the next commands:

```
git clone https://github.com/satnogs/gr-satnogs.git
cd gr-satnogs
mkdir build
cd build
cmake -DLIB_SUFFIX=64 -DCMAKE_INSTALL_PREFIX=/usr ..
make -j4
sudo make install
sudo sh -c 'echo /usr/lib64 > /etc/ld.so.conf.d/lib64.conf'
sudo ldconfig
```

3. Install satnogs-client

Building from source is outside of the scope of this document, we will use the packaged install for now.

Step 3.1: Run the following command to install the packaged version of SatNOGS Client:

```
sudo pip install satnogsclient==0.3
```

4. Configure satnogs-client

SatNOGS Client needs some configuration before running:

Step 4.1: Create a .env file in your home directory (*~/.env*) and add station's details as they are defined at SatNOGS Network:

```
export SATNOGS_API_TOKEN="1234567890qwertyuiopasdfghjklzxcvbnm1234"
export SATNOGS_STATION_ID="65"
export SATNOGS_STATION_LAT="40.662"
export SATNOGS_STATION_LON="23.337"
export SATNOGS_STATION_ELEV="150"
export SATNOGS_NETWORK_API_URL="https://network-dev.satnogs.org/api/"
```

Step 4.2: There are more option you can export in the created .env file. You will probably need to change the default values of the settings bellow:

SATNOGS_RX_DEVICE

- Defines the sdr device. It could be 'usrpb200' or 'rtlsdr'.
- Default Type: string
- Default Value: 'rtlsdr'

SATNOGS_PPM_ERROR

- Defines PPM error of sdr, check *Finding PPM* for more details on PPM.
- Default Type: integer
- Default Value: 0

Step 4.3: Other optional settings:

SATNOGS_APP_PATH

- Defines the path where the sqlite database will be created.
- Default Type: string
- Default Value: '/tmp/.satnogs'

SATNOGS_OUTPUT_PATH

- Defines the path where the observation data will be saved.
- Default Type: string
- Default Value: '/tmp/.satnogs/data'

SATNOGS_COMPLETE_OUTPUT_PATH

- Defines the path where data will be moved after succesful upload on network.
- Default Type: string
- Default Value: '/tmp/.satnogs/data/complete'

SATNOGS_INCOMPLETE_OUTPUT_PATH

- Defines the path where data will be moved after unsuccesful upload on network.
- Default Type: string
- Default Value: '/tmp/.satnogs/data/incomplete'

SATNOGS_ROT_IP

- Defines IP address where rotctld process listens.
- Default Type: string
- Default Value: '127.0.0.1'

SATNOGS_ROT_PORT

- Defines port where rotctld process listens.
- Default Type: integer
- Default Value: 4533

SATNOGS_RIG_IP

- Defines IP address where rigctld process listens.
- Default Type: string

- Default Value: '127.0.0.1'

SATNOGS_RIG_PORT

- Defines port where rigctld process listens.
- Default Type: integer
- Default Value: 4532

5. Prepare SDR Device

In order to have access and use SDR device you need to follow the next steps for you device:

1. USRP B200

Step 5.1.1: Install uhd package:

```
sudo dnf install -y uhd
```

Step 5.1.2: Download uhd images:

```
sudo /usr/bin/uhd_images_downloader
```

Step 5.1.3: As the access will be only by ssh and not by direct login we are not be able to access SDR device through Access Control List(ACL), so we need to setup the appropriate udev rules by following the next steps:

- Copy udev rules from */usr/lib/udev/rules.d/10-usrp-uhd.rules* to */etc/udev/rules.d/10-usrp-uhd.rules*:

```
sudo cp /usr/lib/udev/rules.d/10-usrp-uhd.rules /etc/udev/rules.d/10-usrp-uhd.
↪rules
```

- Replace ACL reference:

```
sudo sed -i 's/0", ENV{ID_SOFTWARE_RADIO}="1"/6"/g' /etc/udev/rules.d/10-usrp-uhd.
↪rules
```

- Reload udev rules:

```
sudo udevadm control --reload-rules
```

- Confirm access on device by running (without sudo, just as single user):

```
uhd_find_devices
```

- In case you don't have access, make sure that the device is connected and that the created user is member of the *dialout* group by running:

```
groups
```

- If user isn't member of *dialout* group run (replace satnogs with the username of your user):

```
sudo usermod -aG dialout satnogs
```

2. RTL-SDR

Step 5.2.1: As the access will be only by ssh and not by direct login we are not be able to access SDR device through Access Control List(ACL), so we need to setup the appropriate udev rules by following the next steps:

- Copy udev rules from */usr/lib/udev/rules.d/10-rtl-sdr.rules* to */etc/udev/rules.d/10-rtl-sdr.rules*:

```
sudo cp /usr/lib/udev/rules.d/10-rtl-sdr.rules /etc/udev/rules.d/10-rtl-sdr.rules
```

- Replace ACL reference and change group ownership:

```
sudo sed -i 's/0", ENV{ID_SOFTWARE_RADIO}="1"/6"/g' /etc/udev/rules.d/10-rtl-sdr.
↪rules
sudo sed -i 's/rtlsdr/dialout/g' /etc/udev/rules.d/10-rtl-sdr.rules
```

- Reload udev rules:

```
sudo udevadm control --reload-rules
```

- If your rtlsdr device was already plugged in, you will need to unplug it and plug it back in. Otherwise, it is safe to plug it in now.
- In case you don't have access, make sure that the device is connected and that the created user is member of the *dialout* group by running:

```
groups
```

- If user isn't member of *dialout* group run (replace satnogs with the username of your user):

```
sudo usermod -aG dialout satnogs
```

- If you had to take that step, log out and log back in

6. Run satnogs-client

1. Manually

In order to manually run satnogs-client you need to follow the next steps:

Step 6.1.1: Export all the environment variables:

```
source .env
```

Step 6.1.2: Start rotctl daemon(note: given example parameters bellow, you may need to change, add or omit some of them. For a Yaesu G-5500 use -m 601 and -s 9600):

```
rotctld -m 202 -r /dev/ttyACM0 -s 19200 &
```

Step 6.1.3: Run the SatNOGS Client:

```
satnogs-client
```

At this point your client should be fully functional! It will check in with the network URL at a 1 minute interval. You should check your ground station page on the website, the station ID will be in a red box until the station checks in, at which time it will turn green. There are many ways to automate the running and control of satnogs, we will give you 2 options below, supervisord and systemd.

2. Automaticaly with Supervisord

Supervisord is one of the ways to automatically run SatNOGS Client. This is very useful especialy after a power failure or reboot of raspberry pi.

In order to setup supervisord we need to follow the next steps:

Step 6.2.1: Install supervisord:

```
sudo dnf install -y supervisor
```

Step 6.2.2: Automate creating of supervisor directory in /var/log path after boot by running:

```
sudo sh -c 'echo "d     /var/log/supervisor 0750 root  root  3d  -" >> /etc/tmpfiles.
↪d/logdirs.conf'
```

Step 6.2.3: Configure supervisord for rotctld

Open with sudo and your favorite editor and add this into /etc/supervisord.d/rotctld.ini:

```
[program:rotctld]
command=/usr/bin/rotctld <rotctld PARAMETERS>
autostart=true
autorestart=true
user=<USERNAME>
priority=1
stdout_logfile=/var/log/supervisor/rotctld.log
stderr_logfile=/var/log/supervisor/rotctld-error.log
```

Replace <USERNAME> with the username of the user you have created and <rotctld PARAMETERS> with the parameters needed to run rotctl in your case.

Step 6.2.4: Configure supervisord for satnogs-client

Add this into /etc/supervisord.d/satnogs.ini:

```
[program:satnogs]
command=/usr/bin/satnogs-client
autostart=true
autorestart=true
user=<USERNAME>
environment=SATNOGS_NETWORK_API_URL="<URL>",SATNOGS_API_TOKEN="<TOKEN>",SATNOGS_
↪STATION_ID="<ID>",SATNOGS_STATION_LAT="<LATITUDE>",SATNOGS_STATION_LON="<LONGITUDE>
↪",SATNOGS_STATION_ELEV="<ELEVATION>"
stdout_logfile=/var/log/supervisor/satnogs.log
stderr_logfile=/var/log/supervisor/satnogs-error.log
```

Replace <USERNAME> with the username of the user you have created. Replace <...> instances in environment with the values you used in .env file, you can also add in this list any other of the *optional settings*.

Step 6.2.5: Reloading supervisord to get the new configuration:

```
sudo systemctl enable supervisord.service
sudo systemctl start supervisord.service
```

With that rotctld and satnogs-client should have started, you can follow the logs in /var/log/supervisor/.

NOTE: In case that you want to change something in .ini files like satnogs environment variables (url from the dev one to production one), then you will need to run:

```
sudo supervisorctl reload
```

3. Automaticaly with Systemd

Systemd is one of the ways to automatically run SatNOGS Client. This is very useful especialy after a power failure or reboot of raspberry pi.

In order to setup systemd we need to follow the next steps:

Step 6.3.1: Create the script which will initialize and run rotctld and satnogs-client in your home directory (*~/start-satnogs-client.sh*) with the following content:

```
rotctld <rotctld PARAMETERS> &
date >> satnogs-auto.log
source .env
satnogs-client
```

Replace <rotctld PARAMETERS> with the parameters needed to run rotctl in your case.

Step 6.3.2: Create as root the file /lib/systemd/system/satnogs-client.service and add the following content:

```
[Unit]
Description=Satnogs Client
Requires=redis.service
After=redis.service

[Service]
User=<USERNAME>
WorkingDirectory=/home/<USERNAME>/
ExecStart=/bin/bash start-satnogs-client.sh
KillMode=control-group

[Install]
WantedBy=multi-user.target
```

Replace <USERNAME> with the username of the user you have created.

Step 6.3.3: Enable and start satnogs-client.service:

```
sudo systemctl enable satnogs-client.service
sudo systemctl start satnogs-client.service
```

With that rotctld and satnogs-client should have started, you can follow the logs with journactl:

```
journalctl -u satnogs-client.service
```

Use *-f* flag if you want to see the latest updates on logs:

```
journalctl -f -u satnogs-client.service
```

NOTE: In case that you want to change something in start-satnogs-client.sh, make the change and then run:

```
sudo systemctl stop satnogs-client.service
sudo systemctl start satnogs-client.service
```

NOTE: In case that you want to change something in satnogs-client.service, make the change and then run:

```
sudo systemctl daemon-reload
```

Finding PPM

This document assumes at least satnogs-client version 0.2.5, found at https://pypi.python.org/pypi/satnogsclient/0.2.5 or https://github.com/satnogs/satnogs-client

The rtl-sdr dongles are not perfectly tuned and there is always a bit of shift in the crystal used. To calibrate this we need to find PPM. While rtl_test comes with PPM detection now, it is not very accurate on the raspberry pi due to the lack of a real time clock. To find our PPM from the command line we will use Kalibrate which finds the PPM against known GSM frequencies.:

```
sudo apt-get install autoconf libtool libfftw3-dev
git clone https://github.com/steve-m/kalibrate-rtl
cd kalibrate-rtl
./bootstrap
./configure
make
sudo make install
```

Now we run kal to first scan for channels nearby, then picking a channel or two we run kal again to calculate the PPM offset. In the USA scan the GSM850 range, in Europe GSM900:

```
kal -s GSM850

Found 1 device(s):
  0:  Generic RTL2832U OEM

Using device 0: Generic RTL2832U OEM
Found Elonics E4000 tuner
Exact sample rate is: 270833.002142 Hz
kal: Scanning for GSM-850 base stations.
GSM-850:
     chan: 145 (872.6MHz + 39.349kHz)        power: 226138.00
     chan: 151 (873.8MHz + 39.379kHz)        power: 361536.36
     chan: 157 (875.0MHz + 5.441kHz) power: 385795.74
```

Now pick a channel and calibrate against it (note this process may run for a long time):

```
kal -c 151

Found 1 device(s):
  0:  Generic RTL2832U OEM

  Using device 0: Generic RTL2832U OEM
  Found Elonics E4000 tuner
  Exact sample rate is: 270833.002142 Hz
  kal: Calculating clock frequency offset.
  Using GSM-850 channel 151 (873.8MHz)
  average       [min, max]  (range, stddev)
  + 39.943kHz       [39832, 39987]  (155, 33.017464)
  overruns: 0
  not found: 781
  average absolute error: -45.711 ppm
```

In this case, we use -45.711 for our PPM error setting, SATNOGS_PPM_ERROR. Refer to [settings.py](https://github.com/satnogs/satnogs-client/blob/master/satnogsclient/settings.py).

CHAPTER 2

satnogs-network

Installation

Docker Installation

1. **Requirements**

   You will need docker and docker-compose.

2. **Build the containers**

   Clone source code from the repository:

   ```
   $ git clone https://github.com/satnogs/satnogs-network.git
   $ cd satnogs-network
   ```

   Set your environmental variables:

   ```
   $ cp .env-dist .env
   ```

   Start database container:

   ```
   $ docker-compose up -d db
   ```

   Build satnogs-network container:

   ```
   $ docker-compose build web
   ```

   Run the initialize script to populate the database with scheme and demo data:

   ```
   $ docker-compose run web python manage.py initialize
   ```

   Note that the above command requires internet connection, since it fetches Satellite and Transmitter data from SatNOGS-DB

3. **Run it!**

   Run satnogs-network:

   ```
   $ docker-compose up
   ```

   Your satnogs-network development instance is available in localhost:8000. Go hack!

VirtualEnv Installation

Requirements: You will need python, python-virtualenvwrapper, pip and git

1. **Build the environment**

   Clone source code from the repository:

   ```
   $ git clone https://github.com/satnogs/satnogs-network.git
   ```

   Set up the virtual environment. On first run you should create it and link it to your project path.:

   ```
   $ cd satnogs-network
   $ mkvirtualenv satnogs-network -a .
   ```

   Set your environmental variables:

   ```
   $ cp .env-dist .env
   ```

   Activate your python virtual environment:

   ```
   $ workon satnogs-network
   ```

   Install local development requirements:

   ```
   $ (satnogs-network)$ pip install -r requirements/dev.txt
   ```

2. **Database**

   Create, setup and populate the database with demo data:

   ```
   (satnogs-network)$ ./manage.py initialize
   ```

   Note that the above command requires internet connection, since it fetches Satellite and Transmitter data from SatNOGS-DB

3. **Run it!**

   Just run it:

   ```
   (satnogs-network)$ ./manage.py runserver
   ```

   Your satnogs-network development instance is available in localhost:8000. Go hack!

Contribute

Thank you for your interest in contributing to SatNOGS! There are always bugs to file; bugs to fix in code; improvements to be made to the documentation; and more.

The below instructions are for software developers who want to work on satnogs-network code.

Git workflow

When you want to start contributing, you should follow the installation instructions, then...

1. (Optional) Set your cloned fork to track upstream changes (changes to the main repository), then fetch and merge changes from the upstream branch:

```
$ git remote add --track master upstream git://github.com/satnogs/satnogs-network
$ git fetch upstream
$ git merge upstream/master
```

2. Set up a branch for a particular set of changes and switch to it:

```
$ git branch my_branch
$ git checkout my_branch
```

3. Commit changes to the code!
4. Code!
5. Lint the code:

```
$ flake8 network
```

   and fix any errors.

6. Commit changes to the code!
7. When you're done, figure out how many commits you've made:

```
$ git log
```

8. Squash all those commits into a single commit that has a good git commit message. (Example assumes you made 4 commits):

```
$ git rebase -i HEAD~4
```

9. Use the interactive editor that pops up to pick/squash your commits:

```
pick 01d1239 [fix bug 893291] Make it go to 11
squash 32as32p added the library and made some minor changes
squash 30ame3z build the template
squash 91pcla8 ugh fix a semicolon bug in that last commit
```

10. Push your changes to your fork:

```
$ git push origin my_branch
```

11. Issue a pull request on GitHub
12. Wait to hear from one of the core developers

If you're asked to change your commit message, you can amend the message and force commit:

```
$ git commit --amend
$ git push -f origin my_branch
```

If you're asked to make changes on your code you can stage them and amend the commit:

```
$ git add my_changed_files
$ git commit --amend
$ git push -f origin my_branch
```

If you need more Git expertise, a good resource is the Git book.

Templates

satnogs-network uses Django's template engine templates.

Coding Style

1. Four space indentation (no tabs), two whitespace on html documents.
2. Use single quotes for strings. Double quotes used only for html attributes.
3. Keep lines shorter than 100 characters when possible (especially at python code)

Follow the PEP8 and PEP257 Style Guides.

Most important things:

1. Separate top-level function and class definitions with two blank lines.
2. Method definitions inside a class are separated by a single blank line.
3. Use whitespace between comma seperated values.
4. Use white space between assignments and expressions (except parameter values).
5. Don't use whitespace before or after parentheses, brackets or braces.
6. Classes should use CamelCase naming.
7. Functions should use lowercase naming.

What to work on

You can check opened issues. We regurarly open issues for tracking new features. You pick one and start coding.

CHAPTER 3

satnogs-db

Installation

Docker Installation

1. **Requirements**

   You will need docker and docker-compose.

2. **Build the containers**

   Clone source code from the repository:

   ```
   $ git clone https://github.com/satnogs/satnogs-db.git
   $ cd satnogs-db
   ```

   Set your environmental variables:

   ```
   $ cp .env-dist .env
   ```

   Start database containers:

   ```
   $ docker-compose up -d db
   ```

   Build satnogs-db container:

   ```
   $ docker-compose build web
   ```

   Run the initialize script to populate the database with scheme and demo data:

   ```
   $ docker-compose run web python manage.py initialize
   ```

3. **Run it!**

   Run satnogs-db:

```
$ docker-compose up
```

Your satnogs-db development instance is available in localhost:8000. Go hack!

VirtualEnv Installation

Requirements: You will need python, python-virtualenvwrapper, pip and git

1. **Build the environment**

   Clone source code from the repository:

   ```
   $ git clone https://github.com/satnogs/satnogs-db.git
   ```

   Set up the virtual environment. On first run you should create it and link it to your project path.:

   ```
   $ cd satnogs-db
   $ mkvirtualenv satnogs-db -a .
   ```

   Set your environmental variables:

   ```
   $ cp .env-dist .env
   ```

   Activate your python virtual environment:

   ```
   $ workon satnogs-db
   ```

   Install local development requirements:

   ```
   $ (satnogs-db)$ pip install -r requirements/dev.txt
   ```

2. **Database**

   Create, setup and populate the database with demo data:

   ```
   (satnogs-db)$ ./manage.py initialize
   ```

3. **Run it!**

   Just run it:

   ```
   (satnogs-db)$ ./manage.py runserver
   ```

   Your satnogs-db development instance is available in localhost:8000. Go hack!

Contribute

Thank you for your interest in contributing to SatNOGS! There are always bugs to file; bugs to fix in code; improvements to be made to the documentation; and more.

The below instructions are for software developers who want to work on satnogs-db code.

Git workflow

When you want to start contributing, you should follow the installation instructions, then...

1. (Optional) Set your cloned fork to track upstream changes (changes to the main repository), then fetch and merge changes from the upstream branch:

```
$ git remote add --track master upstream git://github.com/satnogs/satnogs-db
$ git fetch upstream
$ git merge upstream/master
```

2. Set up a branch for a particular set of changes and switch to it:

```
$ git branch my_branch
$ git checkout my_branch
```

3. Commit changes to the code!
4. Code!
5. Lint the code and fix any errors:

```
$ flake8 db
```

6. Commit changes to the code!
7. When you're done, figure out how many commits you've made:

```
$ git log
```

8. Squash all those commits into a single commit that has a good git commit message. (Example assumes you made 4 commits):

```
$ git rebase -i HEAD~4
```

9. Use the interactive editor that pops up to pick/squash your commits:

```
pick 01d1239 [fix bug 893291] Make it go to 11
squash 32as32p added the library and made some minor changes
squash 30ame3z build the template
squash 91pcla8 ugh fix a semicolon bug in that last commit
```

10. Push your changes to your fork:

```
$ git push origin my_branch
```

11. Issue a pull request on GitHub
12. Wait to hear from one of the core developers

If you're asked to change your commit message, you can amend the message and force commit:

```
$ git commit --amend
$ git push -f origin my_branch
```

If you're asked to make changes on your code you can stage them and amend the commit:

```
$ git add my_changed_files
$ git commit --amend
$ git push -f origin my_branch
```

If you need more Git expertise, a good resource is the Git book.

Templates

satnogs-db uses Django's template engine templates.

Coding Style

1. Four space indentation (no tabs), two whitespace on html documents.
2. Use single quotes for strings. Double quotes used only for html attributes.
3. Keep lines shorter than 100 characters when possible (especially at python code)

Follow the PEP8 and PEP257 Style Guides.

Most important things:

1. Separate top-level function and class definitions with two blank lines.
2. Method definitions inside a class are separated by a single blank line.
3. Use whitespace between comma seperated values.
4. Use white space between assignments and expressions (except parameter values).
5. Don't use whitespace before or after parentheses, brackets or braces.
6. Classes should use CamelCase naming.
7. Functions should use lowercase naming.

What to work on

You can check opened issues. We regurarly open issues for tracking new features. You pick one and start coding.

API

SatNOGS-DB API is a REST API that provides detailed information about Satellites and Transmitters. This document explains how to use the API to retrieve data for your application.

Using API Data

API access is public to anyone. No form of authentication is required. All API data are freely distributed under the CC BY-SA license.

API Methods

Satellites

The `satellites` method of the SatNOGS DB API returns all Satellites currently used for gathering Transmitters data.

Endpoint

```
https://db.satnogs.org/api/satellites/
```

Examples

Show a specific satellite using its Norad Cat ID:

Request:

```
/api/satellites/25544/?format=json
```

Response:

```
{
    norad_cat_id": 25544,
    "name": "ISS (ZARYA)"
}
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

P

R

S

T

W