Global Alliance
for Genomics & Health

GA4GH Documentation

Release undefined

Global Alliance for Genomics and Health

Jan 27, 2017

Contents

This the documentation for version undefined of the GA4GH reference implementation.

CHAPTER 1

Contents

1.1 Introduction

The Data Working Group of the Global Alliance for Genomics and Health has defined an API to facilitate interoperable exchange of genomic data. This is the the documentation for the reference implementation of the API.

Simplicity/clarity The main goal of this implementation is to provide an easy to understand and maintain implementation of the GA4GH API. Design choices are driven by the goal of making the code as easy to understand as possible, with performance being of secondary importance. With that being said, it should be possible to provide a functional implementation that is useful in many cases where the extremes of scale are not important.

Portability The code is written in Python for maximum portability, and it should be possible to run on any modern computer/operating system (Windows compatibility should be possible, although this has not been tested). Our coding guidelines specify using a subset of Python 3 which is backwards compatible with Python 2 following the current best practices. The project currently does not yet support Python 3, as support for it is lacking in several packages that we depend on. However, our eventual goal is to support both Python 2 and 3.

Ease of use The code follows the Python Packaging User Guide. Specifically, pip is used to handle python package dependencies (see below for details). This provides easy installation of the `ga4gh` reference code across a range of operating systems.

1.2 GA4GH API Demo

In this demo, we'll install a copy of the GA4GH reference implementation and run a local version of the server using some example data. We then run some example queries on this server using various different methods to illustrate the basics of the protocol. The server can, of course, be run on any machine on the network, but for simplicity we assume that the client and the server are running on your local machine during this demo.

The instructions for installation here are not intended to be used in a production deployment. See the *Installation* section for a detailed guide on production installation. To run the demo, you will need a working installation of Python 2.7 and also have virtualenv installed. We also need to have zlib and a few other common libraries installed so that we can build some of the packages that the reference server depends on.

On Debian/Ubuntu, for example, we can install these packages using:

```
$ sudo apt-get install python-dev python-virtualenv zlib1g-dev libxslt1-dev libffi-
↪dev libssl-dev
```

On Fedora 22+ (current), the equivalent would be:

```
$ sudo dnf install python-devel python-virtualenv zlib-devel libxslt-devel openssl-
↪devel
```

First, we create a virtualenv sandbox to isolate the demo from the rest of the system, and then activate it:

```
$ virtualenv ga4gh-env
$ source ga4gh-env/bin/activate
```

Now, install the ga4gh package from the Python package index. This will take some time, as some upstream packages will need to be built and installed.

```
(ga4gh-env) $ pip install ga4gh --pre
```

(Older versions of pip might not recognise the `--pre` argument; if not, it is safe to remove it.)

Now we can download some example data, which we'll use for our demo:

```
(ga4gh-env) $ wget https://github.com/ga4gh/server/releases/download/data/ga4gh-
↪example-data_4.6.tar
(ga4gh-env) $ tar -xvf ga4gh-example-data_4.6.tar
```

After extracting the data, we can then run the `ga4gh_server` application:

```
(ga4gh-env) $ ga4gh_server
* Running on http://127.0.0.1:8000/ (Press CTRL+C to quit)
* Restarting with stat
```

(The server is using a default configuration which assumes the existence of the `ga4gh-example-data` directory for simplicity here; see the *Configuration* section for detailed information on how we configure the server.) We now have a server running in the foreground. When it receives requests, it will print out log entries to the terminal. A summary of the server's configuration and data is available in HTML format at `http://locahost:8000`, which can be viewed in a web browser. Leave the server running and open another terminal to complete the rest of the demo.

To try out the server, we must send some requests to it using the GA4GH protocol. One way in which we can do this is to manually create the JSON requests, and send these to the server using cURL:

```
$ curl --data '{}' --header 'Content-Type: application/json' \
http://localhost:8000/datasets/search | jq .
```

In this example, we used the search_datasets method to ask the server for all the Datasets on the server. It responded by sending back some JSON, which we piped into the jq JSON processor to make it easier to read. We get the following result:


```
{
  "nextPageToken": null,
  "datasets": [
    {
      "description": null,
      "name": "1kg-p3-subset",
      "id": "MWtnLXAzLXN1YnNldA=="
    }
```

```
  ]
}
```

In this example we sent a SearchDatasetsRequest object to the server and received a SearchDatasetsResponse object in return. This response object contained one Dataset object, which is contained in the `datasets` array. This approach to interacting with the server is tedious and error prone, as we have to hand-craft the request objects. It is also quite inconvenient, as we may have to request many pages of objects to get all the objects that satisfy our search criteria.

To simplify interacting with the server and to abstract away the low-level network-level details of the server, we provide a client application. To try this out, we start another instance of our virtualenv, and then send the equivalent command using:

```
$ source ga4gh-env/bin/activate
(ga4gh-env) $ ga4gh_client datasets-search http://localhost:8000
```

```
MWtnLXAzLXN1YnNldA==    1kg-p3-subset
```

The output of this command is a summary of the Datasets on that are present on the server. We can also get the output in JSON form such that each object is written on one line:

```
(ga4gh-env) $ ga4gh_client datasets-search -O json http://localhost:8000
```

```
{"description": null, "name": "1kg-p3-subset", "id": "MWtnLXAzLXN1YnNldA=="}
```

This format is quite useful for larger queries, and can be piped into jq to extract fields of interest, pretty printing and so on.

We can perform similar queries for variant data using the search_variants API call. First, we find the IDs of the VariantSets on the server using the search_variant_sets method:

```
(ga4gh-env) $ ga4gh_client variantsets-search http://localhost:8000
```

```
MWtnLXAzLXN1YnNldDptdm5jYWxs    mvncall
```

This tells us that we have one VariantSet on the server, with ID `MWtnLXAzLXN1YnNldDptdm5jYWxs` and name `mvncall`. We can then search for variants overlapping a given interval in a VariantSet as follows:

```
(ga4gh-env) $ ga4gh_client variants-search http://localhost:8000 \
--referenceName=1 --start=45000 --end=50000
```

The output of the client program is a summary of the data received in a free text form. This is not intended to be used as the input to other programs, and is simply a data exploration tool for users. To really *use* our data, we should use a GA4GH client library.

Part of the GA4GH reference implementation is a client library. This makes sending requests to the server and using the responses very easy. For example, to run the same query as we performed above, we can use the following code:

```
from __future__ import print_function

from ga4gh.client import client

httpClient = client.HttpClient("http://localhost:8000")
# Get the datasets on the server.
datasets = list(httpClient.search_datasets())
# Get the variantSets in the first dataset.
variantSets = list(httpClient.search_variant_sets(
```

```
    dataset_id=datasets[0].id))
# Now get the variants in the interval [45000, 50000) on chromosome 1
# in the first variantSet.
iterator = httpClient.search_variants(
    variant_set_id=variantSets[0].id,
    reference_name="1", start=45000, end=50000)
for variant in iterator:
    print(
        variant.reference_name, variant.start, variant.end,
        variant.reference_bases, variant.alternate_bases, sep="\t")
```

If we save this script as `ga4gh-demo.py` we can then run it using:

```
(ga4gh-env) $ python ga4gh-demo.py
```

1.2.1 Host the 1000 Genomes VCF

The GA4GH reference server uses a registry of files and URLs to populate its data repository. In this tutorial we will use the command-line client to create a registry similar to that used by 1kgenomes.ga4gh.org. Your system should have samtools installed, and at least 30GB to host the VCF and reference sets.

Repo administrator CLI

The CLI has methods for adding and removing Feature Sets, Read Group Sets, Variant Sets, etc. Before we can begin adding files we must first initialize an empty registry database. The directory that this database is in should be readable and writable by the current user, as well as the user running the server.

```
$ ga4gh_repo init registry.db
```

This command will create a file `registry.db` in the current working directory. This file should stay relatively small (a few MB for thousands of files).

Now we will add a dataset to the registry, which is a logical container for the genomics data we will later add. You can optionally provide a description using the `--description` flag.

```
$ ga4gh_repo add-dataset registry.db 1kgenomes \
    --description "Variants from the 1000 Genomes project and GENCODE genes␣
↪annotations"
```

Add a Reference Set

It is possible for a server to host multiple reference assemblies. Here we will go through all the steps of downloading and adding the FASTA used for the 1000 Genomes VCF.

```
$ wget ftp://ftp.1000genomes.ebi.ac.uk//vol1/ftp/technical/reference/phase2_reference_
↪assembly_sequence/hs37d5.fa.gz
```

This file is provided in `.gz` format, which we will decompress, and then with samtools installed on the system, recompress it using `bgzip`.

```
$ gunzip hs37d5.fa.gz
$ bgzip hs37d5.fa
```

This may take a few minutes depending on your system as this file is around 3GB. Next, we will add the reference set.

```
$ ga4gh_repo add-referenceset registry.db /full/path/to/hs37d5.fa.gz \
  -d "NCBI37 assembly of the human genome" --ncbiTaxonId 9606 --name NCBI37 \
  --sourceUri "ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/phase2_
↪reference_assembly_sequence/hs37d5.fa.gz"
```

A number of optional command line flags have been added. We will be referring to the name of this reference set `NCBI37` when we later add the variant set.

Add an ontology

Ontologies provide a source for parsing variant annotations, as well as organizing feature types into ontology terms. A sequence ontology instance must be added to the repository to translate ontology term names in sequence and variant annotations to IDs. Sequence ontology definitions can be downloaded from the Sequence Ontology site.

```
$ wget https://raw.githubusercontent.com/The-Sequence-Ontology/SO-Ontologies/master/
↪so-xp-dec.obo
$ ga4gh_repo add-ontology registry.db /full/path/to/so-xp.obo -n so-xp
```

Add sequence annotations

The GENCODE Genes dataset provides annotations for features on the reference assembly. The server uses a custom storage format for sequence annotations, you can download a prepared set here. It can be added to the registry using the following command. Notice we have told the registry to associate the reference set added above with these annotations.

```
$ wget https://ga4ghstore.blob.core.windows.net/testing/gencode_v24lift37.db
$ ga4gh_repo add-featureset registry.db 1kgenomes /full/path/to/gencode.v24lift37.
↪annotation.db \
    --referenceSetName NCBI37 --ontologyName so-xp
```

Add the 1000 Genomes VCFs

The 1000 Genomes are publicly available on the EBI server. This command uses `wget` to download the "release" VCFs to a directory named release.

```
$ wget -m ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/ -nd -P release -
↪l 1
$ rm release/ALL.wgs.phase3_shapeit2_mvncall_integrated_v5b.20130502.sites.vcf.gz
```

These files are already compressed and indexed. For the server to make use of the files in this directory we must move the *wgs* file, since it covers chromosomes that are represented elsewhere and overlapping VCF are not currently supported. This file could be added as a separate variant set.

We can now add the directory to the registry using the following command. Again, notice we have referred to the reference set by name.

```
$ ga4gh_repo add-variantset registry.db 1kgenomes /full/path/to/release/ \
    --name phase3-release --referenceSetName NCBI37
```

Add a BAM as a Read Group Set

Read Group Sets are the logical equivalent to BAM files within the server. We will add a BAM hosting by the 1000 Genomes S3 bucket. We will first download the index and then add it to the registry.

```
$ wget http://s3.amazonaws.com/1000genomes/phase3/data/HG00096/alignment/HG00096.
↪mapped.ILLUMINA.bwa.GBR.low_coverage.20120522.bam.bai
$ ga4gh_repo add-readgroupset registry.db 1kgenomes \
    -I HG00096.mapped.ILLUMINA.bwa.GBR.low_coverage.20120522.bam.bai \
    --referenceSetName NCBI37 \
    http://s3.amazonaws.com/1000genomes/phase3/data/HG00096/alignment/HG00096.mapped.
↪ILLUMINA.bwa.GBR.low_coverage.20120522.bam \
```

This might take a moment as some metadata about the file will be retrieved from S3.

Start the server

Assuming you have set up your server to run using the registry file just created, you can now start or restart the server to see the newly added data. If the server is running via apache issue `sudo service apache2 restart`. You can then visit the landing page of the running server to see the newly added data.

1.2.2 Use the client package

If you only want to use the client and don't need the server functionality, there is a seperate pypi package, ga4gh-client, which includes only the client. It is also much quicker to install. To install, simply run:

```
(ga4gh-env) $ pip install --pre ga4gh_client
```

This installs the `ga4gh_client` command line program, which provides identical functionality to the `ga4gh_client` which is installed via the `ga4gh` package:

```
(ga4gh-env) $ ga4gh_client datasets-search http://1kgenomes.ga4gh.org
```

Installing the `ga4gh_client` package also gives you access to the client's libraries for use in your own programs:

```
>>> from ga4gh.client import client
>>> client.HttpClient
<class 'ga4gh_client.client.HttpClient'>
```

For more examples of using the GA4GH client visit this iPython notebook.

1.2.3 OIDC Demonstration

If we want authentication, we must have an OIDC authentication provider. One can be found in `oidc-provider`, and run with the `run.sh` script. We can then use this with the `LocalOidConfig` server configuration. So:

```
$ cd oidc-provider && ./run.sh
```

In another shell on the same machine

```
$ python server_dev.py -c LocalOidConfig
```

Make sure you know the hostname the server is running on. It can be found with

```
$ python -c 'import socket; print socket.gethostname()'
```

With a web browser, go to `https://<server hostname>:<server port>`. You may need to accept the security warnings as there are probably self-signed certificates. You will be taken through an authentication flow. When asked for a username and password, try `upper` and `crust`. You will find yourself back at the ga4gh server homepage. On the homepage will be a 'session token' This is the key to access the server with the client tool as follows:

```
(ga4gh-env) $ ga4gh_client --key <key from homepage> variantsets-search https://
↪localhost:8000/current
MWtnLXAzLXN1YnNldDptdm5jYWxs    mvncall
```

1.3 Installation

This section documents the process of deploying the GA4GH reference server in a production setting. The intended audience is therefore server administrators. If you are looking for a quick demo of the GA4GH API using a local installation of the reference server please check out the *GA4GH API Demo*. If you are looking for instructions to get a development system up and running, then please go to the *Development* section.

1.3.1 Deployment on Apache

To deploy on Apache on Debian/Ubuntu platforms, do the following.

First, we install some basic pre-requisite packages:

```
sudo apt-get install python-dev python-virtualenv zlib1g-dev libxslt1-dev libffi-dev␣
↪libssl-dev libcurl4-openssl-dev
```

Install Apache and mod_wsgi, and enable mod_wsgi:

```
sudo apt-get install apache2 libapache2-mod-wsgi
sudo a2enmod wsgi
```

Create the Python egg cache directory, and make it writable by the `www-data` user:

```
sudo mkdir /var/cache/apache2/python-egg-cache
sudo chown www-data:www-data /var/cache/apache2/python-egg-cache/
```

Create a directory to hold the GA4GH server code, configuration and data. For convenience, we make this owned by the current user (but make sure all the files are world-readable).:

```
sudo mkdir /srv/ga4gh
sudo chown $USER /srv/ga4gh
cd /srv/ga4gh
```

Make a virtualenv, and install the ga4gh package:

```
virtualenv ga4gh-server-env
source ga4gh-server-env/bin/activate
pip install ga4gh
deactivate
```

Download and unpack the example data:

```
wget https://github.com/ga4gh/server/releases/download/data/ga4gh-example-data_4.6.tar
tar -xf ga4gh-example-data_4.6.tar
```

Create the WSGI file at `/srv/ga4gh/application.wsgi` and write the following contents:

```
from ga4gh.server.frontend import app as application
import ga4gh.server.frontend as frontend
frontend.configure("/srv/ga4gh/config.py")
```

Create the configuration file at `/srv/ga4gh/config.py`, and write the following contents:

```
DATA_SOURCE = "/srv/ga4gh/ga4gh-example-data/registry.db"
```

Note that it is expected that the user running the server, *www-data*, have write and read access to the directories containing data files.

(Many more configuration options are available — see the *Configuration* section for a detailed discussion on the server configuration and input data.)

Configure Apache. Note that these instructions are for Apache 2.4 or greater. Edit the file `/etc/apache2/sites-available/000-default.conf` and insert the following contents towards the end of the file (*within* the `<VirtualHost:80>...</VirtualHost>` block):

```
WSGIDaemonProcess ga4gh \
    processes=10 threads=1 \
    python-path=/srv/ga4gh/ga4gh-server-env/lib/python2.7/site-packages \
    python-eggs=/var/cache/apache2/python-egg-cache
WSGIScriptAlias /ga4gh /srv/ga4gh/application.wsgi

<Directory /srv/ga4gh>
    WSGIProcessGroup ga4gh
    WSGIApplicationGroup %{GLOBAL}
    Require all granted
</Directory>
```

> **Warning:** Be sure to keep the number of threads limited to 1 in the WSGIDaemonProcess setting. Performance tuning should be done using the processes setting.

The instructions for configuring Apache 2.2 (on Ubuntu 14.04) are the same as above with thee following exceptions:

You need to edit `/etc/apache2/sites-enabled/000-default`

instead of `/etc/apache2/sites-enabled/000-default.conf`

And while in that file, you need to set permissions for the directory to

```
Allow from all
```

instead of

```
Require all granted
```

Now restart Apache:

```
sudo service apache2 restart
```

We will now test to see the server started properly by requesting the landing page.

```
curl http://localhost/ga4gh/ --silent | grep GA4GH
#         <title>GA4GH reference server 0.2.3.dev4+nge0b07f3</title>
#    <h2>GA4GH reference server 0.2.3.dev4+nge0b07f3</h2>
# Welcome to the GA4GH reference server landing page! This page describes
```

We can also test the server by running some API commands. Please refer to the instructions in the *GA4GH API Demo* for how to access data made available by this server.

There are any number of different ways in which we can set up a WSGI application under Apache, which may be preferable in different installations. (In particular, the Apache configuration here may be specific to Ubuntu 14.04, where this was tested.) See the mod_wsgi documentation for more details. These instructions are also specific to Debian/Ubuntu and different commands and directory structures will be required on different platforms.

The server can be deployed on any WSGI compliant web server. See the instructions in the Flask documentation for more details on how to deploy on various other servers.

Troubleshooting

Server errors will be output to the web server's error log by default (in Apache on Debian/Ubuntu, for example, this is `/var/log/apache2/error.log`). Each client request will be logged to the web server's access log (in Apache on Debian/Ubuntu this is `/var/log/apache2/access.log`).

For more server configuration options see *Configuration*

1.3.2 Deployment on Docker

It is also possible to deploy the server using Docker.

First, you need an environment running the docker daemon. For non-production use, we recommend boot2docker. For production use you should install docker on a stable linux distro. Please reference the platform specific Docker installation instructions. OSX and Windows are instructions for boot2docker.

Local Dataset Mounted as Volume

If you already have a dataset on your machine, you can download and deploy the apache server in one command:

```
docker run -e GA4GH_DATA_SOURCE=/data -v /my/ga4gh_data/:/data:ro -d -p 8000:80 --
↪name ga4gh_server ga4gh/server:latest
```

Replace `/my/ga4gh_data/` with the path to your data.

This will:

- pull the automatically built image from Dockerhub
- start an apache server running mod_wsgi on container port 80
- mount your data read-only to the docker container
- assign a name to the container
- forward port 8000 to the container.

For more information on docker run options, see the run reference.

Demo Dataset Inside Container

If you do not have a dataset yet, you can deploy a container which includes the demo data:

```
docker run -d -p 8000:80 --name ga4gh_demo ga4gh/server:latest
```

This is identical to the production container, except that a copy of the demo data is included and appropriate defaults are set.

Developing Client Code: Run a Client Container and a Server

In this example you run a server as a daemon in one container, and the client as an ephemeral instance in another container. From the client, the server is accessible at `http://server/`, and the `/tmp/mydev` directory is mounted at `/app/mydev/`. Any changes you make to scripts in `mydev` will be reflected on the host and container and persist even after the container dies.

```
# make a development dir and place the example client script in it
mkdir /tmp/mydev
curl https://raw.githubusercontent.com/ga4gh/server/master/scripts/demo_example.py > /
↪tmp/mydev/demo_example.py
chmod +x /tmp/mydev/demo_example.py

# start the server daemon
# assumes the demo data on host at /my/ga4gh_data
docker run -e GA4GH_DEBUG=True -e GA4GH_DATA_SOURCE=/data -v /my/ga4gh_data/:/data:ro␣
↪-d --name ga4gh_server ga4gh/server:latest

# start the client and drop into a bash shell, with mydev/ mounted read/write
# --link adds a host entry for server, and --rm destroys the container when you exit
docker run -e GA4GH_DEBUG=True -v /tmp/mydev/:/app/mydev:rw -it --name ga4gh_client --
↪link ga4gh_server:server --entrypoint=/bin/bash --rm ga4gh/server:latest

# call the client code script
root@md5:/app# ./mydev/demo_example.py

# call the command line client
root@md5:/app# ga4gh_client variantsets-search http://server/current

#exit and destroy the client container
root@md5:/app# exit
```

Ports

The `-p 8000:80` argument to `docker run` will run the docker container in the background, and translate calls from your host environment port 8000 to the docker container port 80. At that point you should be able to access it like a normal website, albeit on port 8000. Running in boot2docker, you will need to forward the port from the boot2docker VM to the host. From a terminal on the host to forward traffic from localhost:8000 to the VM 8000 on OSX:

```
VBoxManage controlvm boot2docker-vm natpf1 "ga4gh,tcp,127.0.0.1,8000,,8000"
```

For more info on port forwarding see the VirtualBox manual and this wiki article.

Advanced

If you want to build the images yourself, that is possible. The ga4gh/server repo builds automatically on new commits, so this is only needed if you want to modify the Dockerfiles, or build from a different source.

The prod and demo builds are based off of mod_wsgi-docker, a project from the author of mod_wsgi. Please reference the Dockerfiles and documentation for that project during development on these builds.

Examples

Build the code at server/ and run for production, serving a dataset on local host located at `/my/dataset`

```
cd server/
docker build -t my-repo/my-image .
docker run -e GA4GH_DATA_SOURCE=/dataset -v /my/dataset:/dataset:ro -itd -p 8000:80 --
↪name ga4gh_server my-repo/my-image
```

Build and run the production build from above, with the demo dataset in the container (you will need to modify the FROM line in `/deploy/variants/demo/Dockerfile` if you want to use your image from above as the base):

Troubleshooting Docker

DNS

The docker daemon's DNS may be corrupted if you switch networks, especially if run in a VM. For boot2docker, running udhcpc on the VM usually fixes it. From a terminal on the host:

```
eval "$(boot2docker shellinit)"
boot2docker ssh
>     sudo udhcpc
(password is tcuser)
```

DEBUG

To enable DEBUG on your docker server, call docker run with `-e GA4GH_DEBUG=True`

```
docker run -itd -p 8000:80 --name ga4gh_demo -e GA4GH_DEBUG=True ga4gh/server:latest
```

This will set the environment variable which is read by config.py

You can then get logs from the docker container by running `docker logs (container)` e.g. `docker logs ga4gh_demo`

1.3.3 Installing the development version on Mac OS X

Prerequisites

First install libraries and header code for Python 2.7. It will be a lot easier if you have Homebrew, the "missing package manager" for OS X, installed first. To install Homebrew, paste the following at a Terminal prompt ($):

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"
```

Now use `brew install` to install Python if you don't have Python 2.7 installed and then `pip install`, which comes with Python, can be used to install virtual environment:

```
brew install python
pip install virtualenv
```

Install

Download source code from GitHub to the project target folder, here assumed to be `~/ga4gh`: (If you haven't already done so, set up github to work from your command line.)

```
git clone https://github.com/ga4gh/server.git
```

Before installing Python library dependencies, create a virtualenv sandbox to isolate it from the rest of the system, and then activate it:

```
cd server
virtualenv ga4gh-env
source ga4gh-env/bin/activate
```

Install Python dependencies:

```
pip install -r dev-requirements.txt -c constraints.txt
```

Test and run

Run tests to verify the install:

```
ga4gh_run_tests
```

Please refer to the instructions in the *GA4GH API Demo* for how to access data made available by this server.

1.4 Configuration

The GA4GH reference server *Configuration file*. allows Flask and application specific configuration values to be set.

1.4.1 Configuration file

The GA4GH reference server is a Flask application and uses the standard Flask configuration file mechanisms. Many configuration files will be very simple, and will consist of just one directive instructing the server where to find the data repository; example, we might have

```
DATA_SOURCE = "/path/to/registry.db"
```

For production deployments, we shouldn't need to add any more configuration than this, as the other keys have sensible defaults. However, all of Flask's builtin configuration values are supported, as well as the extra custom configuration values documented here. For information on preparing a data source see *Data repository* .

When debugging deployment issues, it can be very useful to turn on extra debugging information as follows:

```
DEBUG = True
```

Warning: Debugging should only be used temporarily and not left on by default. Running the server with Flask debugging enable is insecure and should never be used in a production environment.

Configuration Values

DEFAULT_PAGE_SIZE The default maximum number of values to fill into a page when responding to search queries. If a client does not specify a page size in a query, this value is used.

MAX_RESPONSE_LENGTH The approximate maximum size of the server buffer used when creating responses. This is somewhat smaller than the size of the JSON response returned to the client. When a client makes a search request with a given page size, the server will process this query and incrementally build a response until (a) the number of values in the page list is equal to the page size; (b) the size of the internal buffer in bytes is >= MAX_RESPONSE_LENGTH; or (c) there are no more results left in the query.

REQUEST_VALIDATION Set this to True to strictly validate all incoming requests to ensure that they conform to the protocol. This may result in clients with poor standards compliance receiving errors rather than the expected results.

LANDING_MESSAGE_HTML The server provides a simple landing page at its root. By setting this value to point at a file containing an HTML block element it is possible to customize the landing page. This can be helpful to provide support links or details about the hosted datasets.

OIDC_PROVIDER If this value is provided, then OIDC is configured and SSL is used. It is the URI of the OpenID Connect provider, which should return an OIDC provider configuration document.

OIDC_REDIRECT_URI The URL of the redirect URI for OIDC. This will be something like `https://SERVER_NAME:PORT/oauth2callback`. During testing (and particularly in automated tests), if TESTING is True, we can have this automatically configured, but this is discouraged in production, and fails if TESTING is not True.

OIDC_CLIENT_ID, OIDC_CLIENT_SECRET These are the client id and secret arranged with the OIDC provider, if client registration is manual (google, for instance). If the provider supports automated registration they are not required or used.

OIDC_AUTHZ_ENDPOINT, OIDC_TOKEN_ENDPOINT, OIDC_TOKEN_REV_ENDPOINT If the authorization provider has no discovery document available, you can set the authorization and token endpoints here.

CACHE_DIRECTORY This directory is used to share data between processes when the server is run in a production environment. It defaults to `/tmp/ga4gh/`.

SECRET_KEY The secret key used by the server to encrypt cookies. Preferably, this should be a long (~24 characters) random string, however any string will work.

AUTH0_ENABLED When set to true, enables authentication via Auth0.

AUTH0_SCOPES These are the login identity providers that an Auth0 application is configured to accept. More about scopes can be seen here.

AUTH0_CALLBACK_URL This configuration value let's Auth0 know which URL to return a session to after authentication. It should match the setting in your Auth0 configuration.

AUTH0_HOST The Auth0 host is the domain under which the Auth0 account is hosted.

AUTH0_CLIENT_ID Each application is authenticated to your Auth0 account using a Client ID and secret. This is available in the Auth0 configuration.

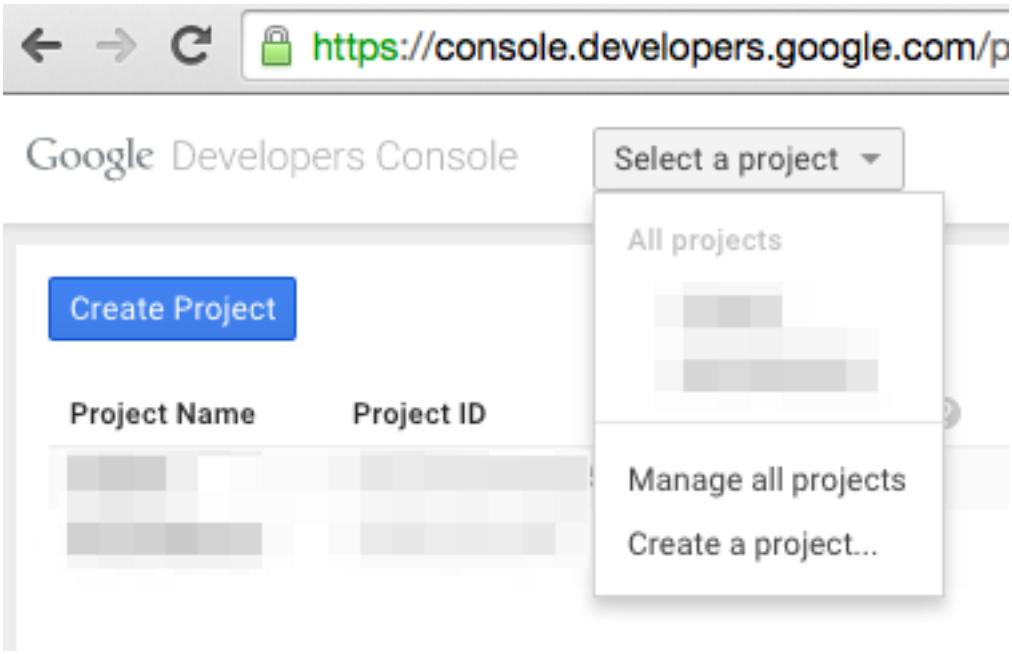
AUTH0_CLIENT_SECRET The client secret is a preshared key between your instance of the server and Auth0 and is available in the Auth0 configuration panel.

AUTHORIZED_EMAILS A comma separated list of user

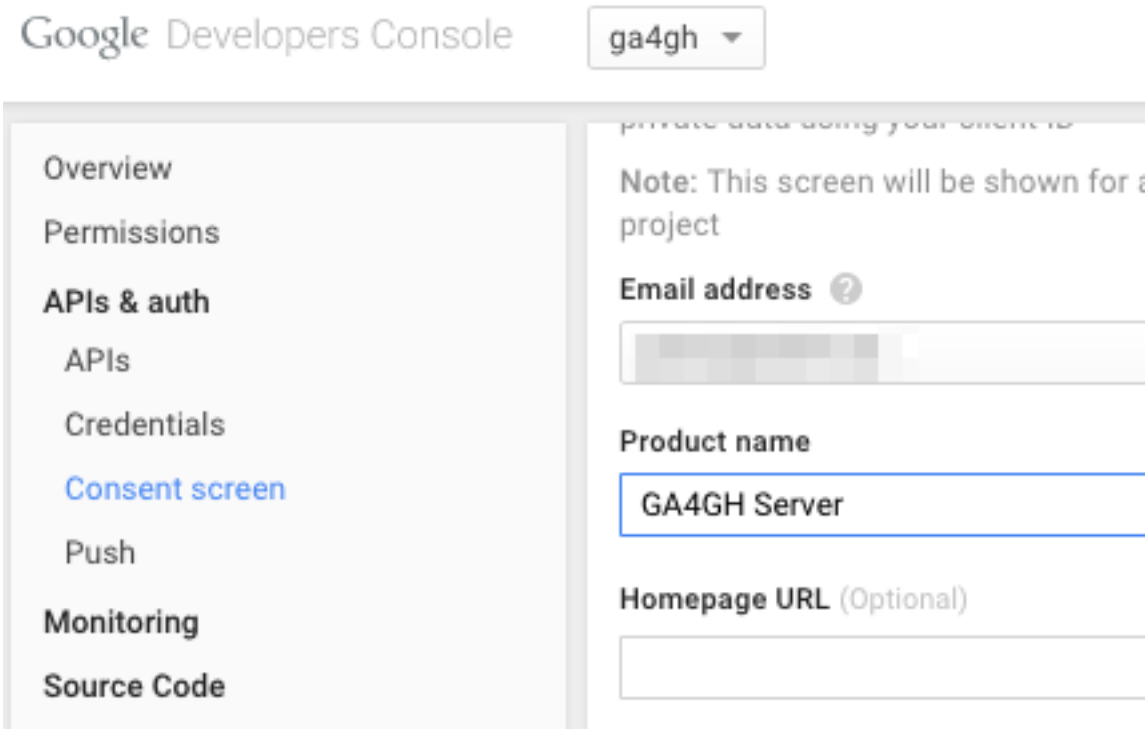
1.4.2 OpenID Connect Providers

The server can be configured to use OpenID Connect (OIDC) for authentication. As an example, here is how one configures it to use Google as the provider.

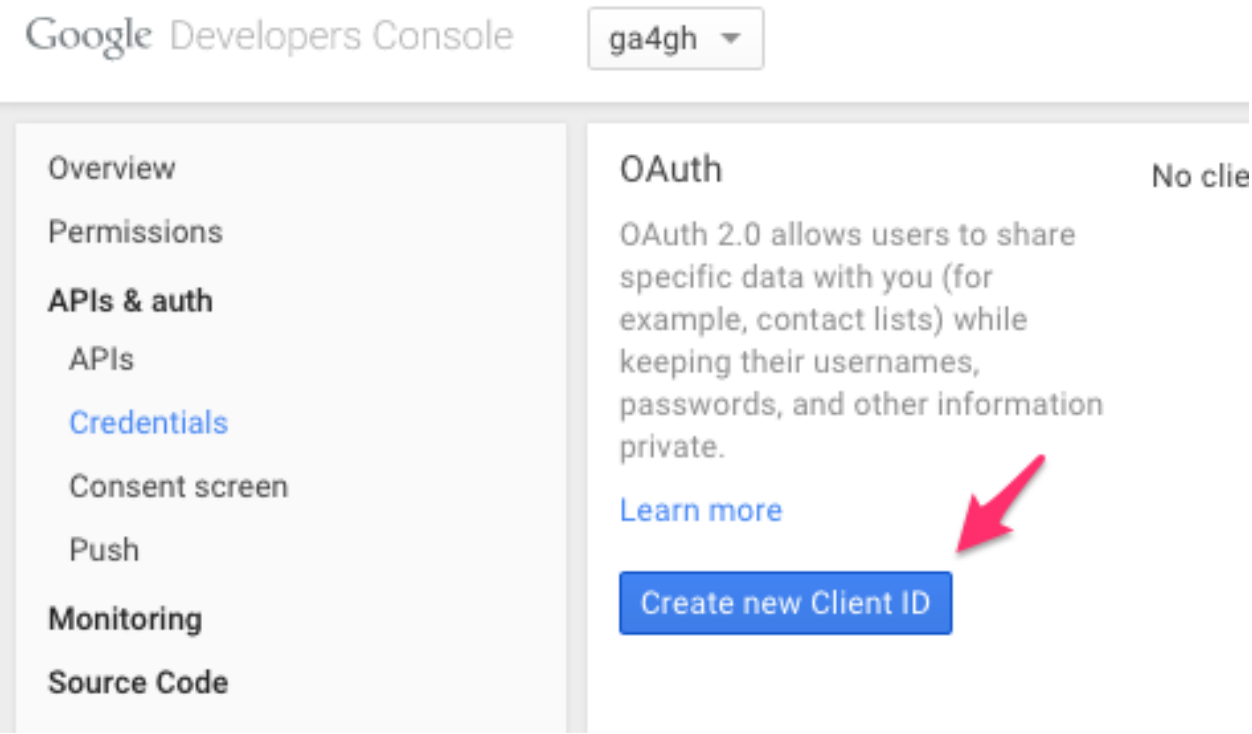
Go to https://console.developers.google.com/project and in create a project.



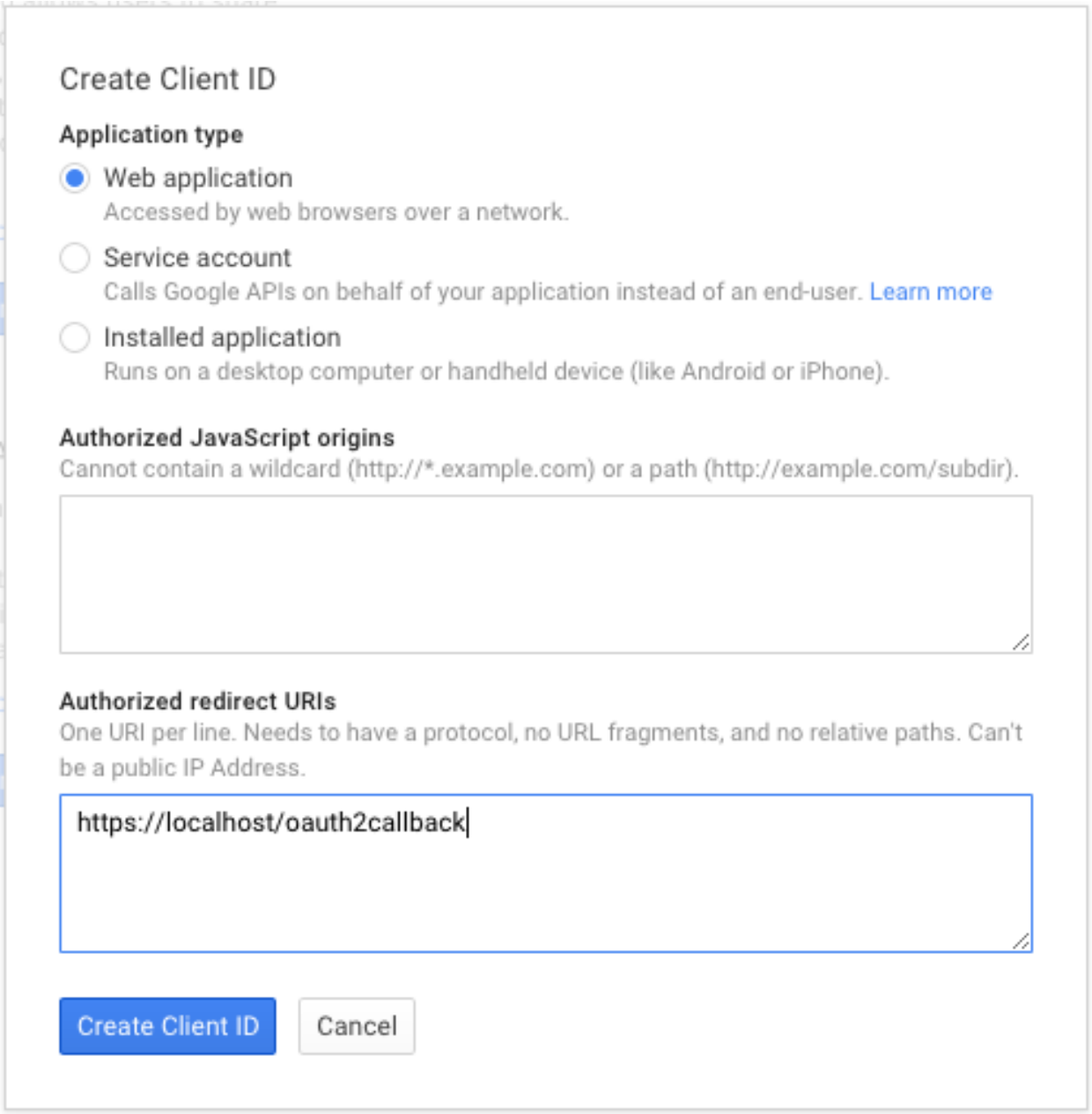
Navigate to the project -> APIs & auth -> Consent Screen and enter a product name



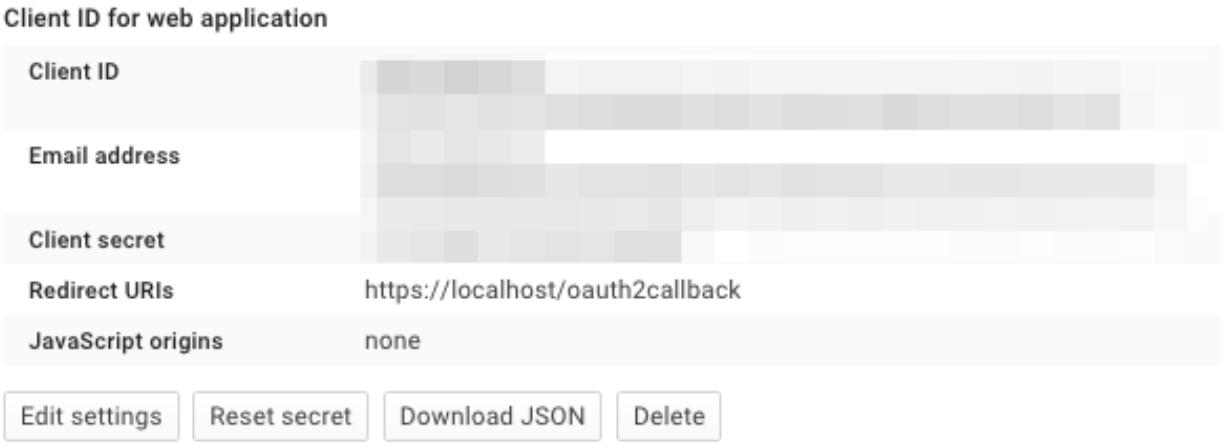
Navigate to project -> APIs & auth -> Credentials, and create a new client ID.



Create the client as follows:



Which will give you the necessary client id and secret. Use these in the OIDC configuration for the GA4GH server, using the *OIDC_CLIENT_ID* and *OIDC_CLIENT_SECRET* configuration variables. The Redirect URI should match the *OIDC_REDIRECT_URI* configuration variable, with the exception that the redirect URI shown at google does not require a port (but the configuration variable does). Finally, set the *SECRET_KEY* to any string for storing cookies.



1.5 Data repository

Each GA4GH server represents a repository of information. This repository consists of the reference sets, datasets, readgroup sets, variant sets etc. in the server's data model and may contain data from many different unrelated projects. The server administrator defines and manages this repository using the `ga4gh_repo` command line interface, which provides commands to manage all of the objects represented by a GA4GH server.

The information about the objects in the GA4GH data model is stored in an SQL database, called the "registry DB" or "registry". The registry DB does not contain the raw bulk data but rather "registers" the information about where the server can find this information and some metadata about the object in question. For example, if we have a variant set that is backed by a single VCF file, the registry DB will contain the path to this file as well as the name of the variant set, the reference set it is defined by, and other information necessary to implement the GA4GH protocol. This registry architecture allows us a lot of flexibility in the sources of data that we can use.

1.5.1 Command Line

init

The `init` command initialises a new registry DB at a given file path. This is the first command that must be issued when creating a new GA4GH repository.

```
usage: ga4gh_repo init [-h] [-f] registryPath
```

Positional arguments:

registryPath	the location of the registry database

Options:

-f, --force	do not prompt for confirmation

Examples:

```
$ ga4gh_repo init registry.db
```

list

The `list` command is used to print the contents of a repository to the screen. It is an essential tool for administrators to understand the structure of the repository that they are managing.

Note: The `list` command is under development and will be much more sophisticated in the future. In particular, the output of this command should improve considerably in the near future.

```
usage: ga4gh_repo list [-h] registryPath
```

Positional arguments:

registryPath the location of the registry database

Examples:

```
$ ga4gh_repo list registry.db
```

verify

The `verify` command is used to check that the integrity of the data in a repository. The command checks each container object in turn and ensures that it can read data from it. Read errors can occur for any number of reasons (for example, a VCF file may have been moved to another location since it was added to the registry), and the `verify` command allows an administrator to check that all is well in their repository.

Note: The `verify` command is under development and will be much more sophisticated in the future. In particular, the output of this command should improve considerably in the near future.

```
usage: ga4gh_repo verify [-h] registryPath
```

Positional arguments:

registryPath the location of the registry database

Examples:

```
$ ga4gh_repo verify registry.db
```

add-dataset

Creates a new dataset in a repository. A dataset is an arbitrary collection of ReadGroupSets, VariantSets, VariantAnnotationSets and FeatureSets. Each dataset has a name, which is used to identify it in the repository manager.

```
usage: ga4gh_repo add-dataset [-h] [-i INFO] [-d DESCRIPTION]
                              registryPath datasetName
```

Positional arguments:

registryPath the location of the registry database

datasetName the name of the dataset

Options:

-i, --info the info of the dataset

-d, --description The human-readable description of the dataset.

Examples:

```
$ ga4gh_repo add-dataset registry.db 1kg -d 'Example dataset using 1000 genomes data'
```

Adds the dataset with the name `1kg` and description `'Example dataset using 1000 genomes data'` to the registry database `registry.db`.

add-referenceset

Adds a reference set derived from a FASTA file to a repository. Each record in the FASTA file will correspond to a Reference in the new ReferenceSet. The input FASTA file must be compressed with `bgzip` and indexed using `samtools faidx`. Each ReferenceSet contains a number of metadata values (.e.g. `ncbiTaxonId`) which can be set using command line options.

```
usage: ga4gh_repo add-referenceset [-h] [-r] [-n NAME] [-d DESCRIPTION]
                                   [--ncbiTaxonId NCBITAXONID]
                                   [--isDerived ISDERIVED]
                                   [--assemblyId ASSEMBLYID]
                                   [--sourceAccessions SOURCEACCESSIONS]
                                   [--sourceUri SOURCEURI]
                                   registryPath filePath
```

Positional arguments:

registryPath the location of the registry database

filePath The path of the FASTA file to use as a reference set. This file must be bgzipped and indexed.

Options:

-r, --relativePath store relative path in database

-n, --name The name of the reference set

-d, --description The human-readable description of the reference set.

--ncbiTaxonId The NCBI Taxon Id

--isDerived Indicates if this reference set is derived from another

--assemblyId The assembly id

--sourceAccessions The source accessions (pass as comma-separated list)

--sourceUri The source URI

Examples:

```
$ ga4gh_repo add-referenceset registry.db hs37d5.fa.gz \
    --description "NCBI37 assembly of the human genome" \
    --ncbiTaxonId 9606 --name NCBI37 \
    --sourceUri ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/phase2_
↪reference_assembly_sequence/hs37d5.fa.gz
```

Adds a reference set used in the 1000 Genomes project using the name `NCBI37`, also setting the `ncbiTaxonId` to 9606 (human).

add-biosample

Adds a new biosample to the repository. The biosample argument is a JSON document according to the GA4GH JSON schema.

```
usage: ga4gh_repo add-biosample [-h]
                                registryPath datasetName biosampleName
                                biosample
```

Positional arguments:

registryPath	the location of the registry database
datasetName	the name of the dataset
biosampleName	the name of the biosample
biosample	the JSON of the biosample

Examples:

```
$ ga4gh_repo add-biosample registry.db dataset1 HG00096 '{"individualId": "abc"}'
```

Adds the biosample named HG00096 to the repository with the individual ID "abc".

add-individual

Adds a new individual to the repository. The individual argument is a JSON document following the GA4GH JSON schema.

```
usage: ga4gh_repo add-individual [-h]
                                 registryPath datasetName individualName
                                 individual
```

Positional arguments:

registryPath	the location of the registry database
datasetName	the name of the dataset
individualName	the name of the individual
individual	the JSON of the individual

Examples:

```
$ ga4gh_repo add-individual registry.db dataset1 HG00096 '{"description": "A␣
↪description"}'
```

add-ontology

Adds a new ontology to the repository. The ontology supplied must be a text file in OBO format. If you wish to serve sequence or variant annotations from a repository, a sequence ontology (SO) instance is required to translate ontology term names held in annotations to ontology IDs. Sequence ontology definitions can be downloaded from the Sequence Ontology site.

```
usage: ga4gh_repo add-ontology [-h] [-r] [-n NAME] registryPath filePath
```

Positional arguments:

registryPath the location of the registry database

filePath The path of the OBO file defining this ontology.

Options:

-r, --relativePath store relative path in database

-n, --name The name of the ontology

Examples:

```
$ ga4gh_repo add-ontology registry.db path/to/so-xp.obo
```

Adds the sequence ontology `so-xp.obo` to the repository using the default naming rules.

add-variantset

Adds a variant set to a named dataset in a repository. Variant sets are currently derived from one or more non-overlapping VCF/BCF files which may be either stored locally or come from a remote URL. Multiple VCF files can be specified either directly on the command line or by providing a single directory argument that contains indexed VCF files. If remote URLs are used then index files in the local file system must be provided using the `-I` option.

```
usage: ga4gh_repo add-variantset [-h] [-r] [-I indexFiles [indexFiles ...]]
                                 [-n NAME] [-R REFERENCESETNAME]
                                 [-O ONTOLOGYNAME] [-a]
                                 registryPath datasetName dataFiles
                                 [dataFiles ...]
```

Positional arguments:

registryPath the location of the registry database

datasetName the name of the dataset

dataFiles The VCF/BCF files representing the new VariantSet. These may be specified either one or more paths to local files or remote URLS, or as a path to a local directory containing VCF files. Either a single directory argument may be passed or a list of file paths/URLS, but not a mixture of directories and paths.

Options:

-r, --relativePath store relative path in database

-I, --indexFiles The index files for the VCF/BCF files provided in the dataFiles argument. These must be provided in the same order as the data files.

-n, --name The name of the VariantSet

-R, --referenceSetName the name of the reference set to associate with this VariantSet

-O, --ontologyName the name of the sequence ontology instance used to translate ontology term names to IDs in this VariantSet

-a, --addAnnotationSets If the supplied VCF file contains annotations, create the corresponding VariantAnnotationSet.

Examples:

```
$ ga4gh_repo add-variantset registry.db 1kg 1kgPhase1/ -R NCBI37
```

Adds a new variant set to the dataset named `1kg` in the repository defined by the registry database `registry.db` using the VCF files contained in the directory `1kgPhase1`. Note that this directory must also contain the corresponding indexes for these files. We associate the reference set named `NCBI37` with this new variant set. Because we do not provide a `--name` argument, a name is automatically generated using the default name generation rules.

```
$ ga4gh_repo add-variantset registry.db 1kg \
    1kgPhase1/chr1.vcf.gz 1kg/chr2.vcf.gz -n phase1-subset -R NCBI37
```

Like the last example, we add a new variant set to the dataset `1kg`, but here we only use the VCFs for chromosomes 1 and 2. We also specify the name for this new variant set to be `phase1-subset`.

```
$ ga4gh_repo add-variantset registry.db 1kg \
    --name phase1-subset-remote -R NCBI37 \
    --indexFiles ALL.chr1.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf.gz.
↪tbi ALL.chr2.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf.gz.tbi \
    ftp://ftp.ncbi.nlm.nih.gov/1000genomes/ftp/release/20110521/ALL.chr1.phase1_
↪release_v3.20101123.snps_indels_svs.genotypes.vcf.gz \
    ftp://ftp.ncbi.nlm.nih.gov/1000genomes/ftp/release/20110521/ALL.chr2.phase1_
↪release_v3.20101123.snps_indels_svs.genotypes.vcf.gz
```

This example performs the same task of creating a subset of the phase1 VCFs, but this time we use the remote URL directly and do not keep a local copy of the VCF file. Because we are using remote URLs to define the variant set, we have to download a local copy of the corresponding index files and provide them on the command line using the `--indexFiles` option.

add-readgroupset

Adds a readgroup set to a named dataset in a repository. Readgroup sets are currently derived from a single indexed BAM file, which can be either stored locally or based on a remote URL. If the readgroup set is based on a remote URL, then the index file must be stored locally and specified using the `--indexFile` option.

Each readgroup set must be associated with the reference set that it is aligned to. The `add-readgroupset` command first examines the headers of the BAM file to see if it contains information about references, and then looks for a reference set with name equal to the genome assembly identifer defined in the header. (Specifically, we read the `@SQ` header line and use the value of the `AS` tag as the default reference set name.) If this reference set exists, then the readgroup set will be associated with it automatically. If it does not (or we cannot find the appropriate information in the header), then the `add-readgroupset` command will fail. In this case, the user must provide the name of the reference set using the `--referenceSetName` option.

```
usage: ga4gh_repo add-readgroupset [-h] [-n NAME] [-R REFERENCESETNAME] [-r]
                                   [-I INDEXFILE]
                                   registryPath datasetName dataFile
```

Positional arguments:

registryPath	the location of the registry database
datasetName	the name of the dataset
dataFile	The file path or URL of the BAM file for this ReadGroupSet

Options:

-n, --name	The name of the ReadGroupSet
-R, --referenceSetName	the name of the reference set to associate with this ReadGroupSet
-r, --relativePath	store relative path in database

-I, --indexFile The file path of the BAM index for this ReadGroupSet. If the dataFile argument is a local file, this will be automatically inferred by appending '.bai' to the file name. If the dataFile is a remote URL the path to a local file containing the BAM index must be provided

Examples:

```
$ ga4gh_repo add-readgroupset registry.db 1kg \
    path/to/HG00114.chrom11.ILLUMINA.bwa.GBR.low_coverage.20120522.bam
```

Adds a new readgroup set for an indexed 1000 Genomes BAM file stored on the local file system. The index file follows the usual convention and is stored in the same directory as the BAM file and has an extra `.bai` extension. The name of the readgroup set is automatically derived from the file name, and the reference set automatically set from the BAM header.

```
$ ga4gh_repo add-readgroupset registry.db 1kg ga4gh-example-data/HG00096.bam \
    -R GRCh37-subset -n HG0096-subset
```

Adds a new readgroup set based on a subset of the 1000 genomes reads for the HG00096 sample from the example data used in the reference server. In this case we specify that the reference set name `GRCh37-subset` be associated with the readgroup set. We also override the default name generation rules and specify the name `HG00096-subset` for the new readgroup set.

```
$ ga4gh_repo add-readgroupset registry.db 1kg \
    -n HG00114-remote
    -I /path/to/HG00114.chrom11.ILLUMINA.bwa.GBR.low_coverage.20120522.bam.bai
    ftp://ftp.ncbi.nlm.nih.gov/1000genomes/ftp/phase3/data/HG00114/alignment/HG00114.
↪chrom11.ILLUMINA.bwa.GBR.low_coverage.20120522.bam
```

Adds a new readgroups set based on a 1000 genomes BAM directly from the NCBI FTP server. Because this readgroup set uses a remote FTP URL, we must specify the location of the `.bai` index file on the local file system.

init-rnaquantificationset

Initializes a rnaquantification set.

```
usage: ga4gh_repo init-rnaquantificationset [-h] registryPath filePath
```

Positional arguments:

registryPath the location of the registry database

filePath The path to the resulting Quantification Set

Examples:

```
$ ga4gh_repo init-rnaquantificationset repo.db rnaseq.db
```

Initializes the RNA Quantification Set with the filename rnaseq.db.

add-rnaquantification

Adds a rnaquantification to a RNA quantification set.

RNA quantification formats supported are currently kallisto and RSEM.

```
usage: ga4gh_repo add-rnaquantification [-h] [--biosampleName BIOSAMPLENAME]
                                        [--readGroupSetName READGROUPSETNAME]
                                        [--featureSetNames FEATURESETNAMES]
                                        [-n NAME] [-d DESCRIPTION] [-t]
                                        filePath quantificationFilePath format
                                        registryPath datasetName
```

Positional arguments:

filePath	The path to the RNA SQLite database to create or modify
quantificationFilePath	The path to the expression file.
format	format of the quantification input data
registryPath	the location of the registry database
datasetName	the name of the dataset

Options:

--biosampleName	Biosample Name
--readGroupSetName	Read Group Set Name
--featureSetNames	Comma separated list
-n, --name	The name of the rna quantification
-d, --description	The human-readable description of the RnaQuantification.
-t, --transcript	sets the quantification type to transcript

Examples:

```
$ ga4gh_repo add-rnaquantification rnaseq.db data.tsv \
         kallisto ga4gh-example-data/registry.db brca1 \
        --biosampleName HG00096 --featureSetNames gencodev19
        --readGroupSetName HG00096rna --transcript
```

Adds the data.tsv in kallisto format to the *rnaseq.db* quantification set with optional fields for associating a quantification with a Feature Set, Read Group Set, and Biosample.

add-rnaquantificationset

When the desired RNA quantification have been added to the set, use this command to add them to the registry.

```
usage: ga4gh_repo add-rnaquantificationset [-h] [-R REFERENCESETNAME]
                                           [-n NAME]
                                           registryPath datasetName filePath
```

Positional arguments:

registryPath	the location of the registry database
datasetName	the name of the dataset
filePath	The path to the converted SQLite database containing RNA data

Options:

-R, --referenceSetName	the name of the reference set to associate with this RnaQuantification-Set

-n, --name The name of the RnaQuantificationSet

Examples:

```
$ ga4gh_repo add-rnaquantificationset registry.db brca1 rnaseq.db \
    -R hg37 -n rnaseq
```

Adds the RNA quantification set *rnaseq.db* to the registry under the *brca1* dataset. The flags set the reference genome to be hg37 and the name of the set to *rnaseq*.

add-phenotypeassociationset

Adds an rdf object store. The cancer genome database Clinical Genomics Knowledge Base http://nif-crawler.neuinfo.org/monarch/ttl/cgd.ttl, published by the Monarch project, is the supported format for Evidence.

```
usage: ga4gh_repo add-phenotypeassociationset [-h] [-n NAME]
                                              registryPath datasetName dirPath
```

Positional arguments:

registryPath the location of the registry database

datasetName the name of the dataset

dirPath The path of the directory containing ttl files.

Options:

-n, --name The name of the PhenotypeAssociationSet

Examples:

```
$ ga4gh_repo add-phenotypeassociationset registry.db dataset1 /monarch/ttl/cgd.ttl -n␣
↪cgd
```

remove-dataset

Removes a dataset from the repository and recursively removes all objects (ReadGroupSets, VariantSets, etc) within this dataset.

```
usage: ga4gh_repo remove-dataset [-h] [-f] registryPath datasetName
```

Positional arguments:

registryPath the location of the registry database

datasetName the name of the dataset

Options:

-f, --force do not prompt for confirmation

Examples:

```
$ ga4gh_repo remove-dataset registry.db dataset1
```

Deletes the dataset with name `dataset1` from the repository represented by `registry.db`

remove-referenceset

Removes a reference set from the repository. Attempting to remove a reference set that is referenced by other objects in the repository will result in an error.

```
usage: ga4gh_repo remove-referenceset [-h] [-f] registryPath referenceSetName
```

Positional arguments:

registryPath	the location of the registry database
referenceSetName	the name of the reference set

Options:

-f, --force	do not prompt for confirmation

Examples:

```
$ ga4gh_repo remove-referenceset registry.db NCBI37
```

Deletes the reference set with name `NCBI37` from the repository represented by `registry.db`

remove-biosample

Removes a biosample from the repository.

```
usage: ga4gh_repo remove-biosample [-h] [-f]
                                   registryPath datasetName biosampleName
```

Positional arguments:

registryPath	the location of the registry database
datasetName	the name of the dataset
biosampleName	the name of the biosample

Options:

-f, --force	do not prompt for confirmation

Examples:

```
$ ga4gh_repo remove-biosample registry.db dataset1 HG00096
```

Deletes the biosample with name `HG00096` in the dataset `dataset1` from the repository represented by `registry.db`

remove-individual

Removes an individual from the repository.

```
usage: ga4gh_repo remove-individual [-h] [-f]
                                    registryPath datasetName individualName
```

Positional arguments:

registryPath	the location of the registry database
datasetName	the name of the dataset

individualName the name of the individual

Options:

-f, --force do not prompt for confirmation

Examples:

```
$ ga4gh_repo remove-individual registry.db dataset1 HG00096
```

Deletes the individual with name `HG00096` in the dataset `dataset1` from the repository represented by `registry.db`

remove-ontology

Removes an ontology from the repository. Attempting to remove an ontology that is referenced by other objects in the repository will result in an error.

```
usage: ga4gh_repo remove-ontology [-h] [-f] registryPath ontologyName
```

Positional arguments:

registryPath the location of the registry database

ontologyName the name of the ontology

Options:

-f, --force do not prompt for confirmation

Examples:

```
$ ga4gh_repo remove-ontology registry.db so-xp
```

Deletes the ontology with name `so-xp` from the repository represented by `registry.db`

remove-variantset

Removes a variant set from the repository. This also deletes all associated call sets and variant annotation sets from the repository.

```
usage: ga4gh_repo remove-variantset [-h] [-f]
                                    registryPath datasetName variantSetName
```

Positional arguments:

registryPath the location of the registry database

datasetName the name of the dataset

variantSetName the name of the variant set

Options:

-f, --force do not prompt for confirmation

Examples:

```
$ ga4gh_repo remove-variantset registry.db dataset1 phase3-release
```

Deletes the variant set named `phase3-release` from the dataset named `dataset1` from the repository represented by `registry.db`.

remove-readgroupset

Removes a read group set from the repository.

```
usage: ga4gh_repo remove-readgroupset [-h] [-f]
                                      registryPath datasetName
                                      readGroupSetName
```

Positional arguments:

registryPath	the location of the registry database
datasetName	the name of the dataset
readGroupSetName	the name of the read group set

Options:

-f, --force	do not prompt for confirmation

Examples:

```
$ ga4gh_repo remove-readgroupset registry.db dataset1 HG00114
```

Deletes the readgroup set named `HG00114` from the dataset named `dataset1` from the repository represented by `registry.db`.

remove-rnaquantificationset

Removes a rna quantification set from the repository.

```
usage: ga4gh_repo remove-rnaquantificationset [-h] [-f]
                                              registryPath datasetName
                                              rnaQuantificationSetName
```

Positional arguments:

registryPath	the location of the registry database
datasetName	the name of the dataset
rnaQuantificationSetName	the name of the RNA Quantification Set

Options:

-f, --force	do not prompt for confirmation

Examples:

```
$ ga4gh_repo remove-rnaquantificationset registry.db dataset1 ENCFF305LZB
```

Deletes the rnaquantification set named `ENCFF305LZB` from the dataset named `dataset1` from the repository represented by `registry.db`.

remove-phenotypeassociationset

Removes an rdf object store.

```
usage: ga4gh_repo remove-phenotypeassociationset [-h] [-f]
                                                 registryPath datasetName name
```

Positional arguments:

registryPath	the location of the registry database
datasetName	the name of the dataset
name	The name of the phenotype association set

Options:

-f, --force	do not prompt for confirmation

Examples:

```
$ ga4gh_repo remove-phenotypeassociationset registry.db dataset1  cgd
```

1.6 Development

Thanks for your interest in helping us develop the GA4GH reference implementation! There are lots of ways to contribute, and it's easy to get up and running. This page should provide the basic information required to get started; if you encounter any difficulties please let us know

Warning: This guide is a work in progress, and is incomplete.

1.6.1 Development environment

We need a development Python 2.7 installation, Git, and some basic libraries. On Debian or Ubuntu, we can install these using

```
sudo apt-get install python-dev git zlib1g-dev libxslt1-dev libffi-dev libssl-dev
```

Note: Instructions for configuring the reference server on Mac OS X can be found here *Installation*.

If you don't have admin access to your machine, please contact your system administrator, and ask them to install the development version of Python 2.7 and the development headers for zlib.

You will also need to install Protocol Buffers 3.0 in your development environment. The general process for doing the install is best described in the protobuf documentation here: https://github.com/google/protobuf If you are working on Mac OS X then there is an easy install process through homebrew:

```
brew update && brew install --devel protobuf
```

Once these basic prerequisites are in place, we can then bootstrap our local Python installation so that we have all of the packages we require and we can keep them up to date. Because we use the functionality of the recent versions of `pip` and other tools, it is important to use our own version of it and not any older versions that may be already on the system.

```
wget https://bootstrap.pypa.io/get-pip.py
python get-pip.py --user
```

This creates a user specific site-packages installation for Python, which is based in your `~/.local` directory. This means that you can now install any Python packages you like without needing to either bother your sysadmin or worry about breaking your system Python installation. To use this, you need to add the newly installed version of `pip` to your `PATH`. This can be done by adding something like

```
export PATH=$HOME/.local/bin:$PATH
```

to your `~/.bashrc` file. (This will be slightly different if you use another shell like `csh` or `zsh`.)

We then need to activate this configuration by logging out, and logging back in. Then, test this by running:

```
pip --version
#pip 6.0.8 from /home/username/.local/lib/python2.7/site-packages (python 2.7)
```

From here, we suggest using virtualenv to manage your python environments. You can install and activate a virtual environment using:

```
pip install virtualenv
virtualenv ga4gh-server-env
source ga4gh-server-env/bin/activate
```

1.6.2 Using Development Constraints

The server uses the GA4GH schemas as a basis for serializing and deserializing data. This means that the server depends on the schemas, and at times a developer will need to point at a specific version of the schemas to reflect a change to the data model.

There is a `constraints.txt` file in the root of the source tree that can be used to pin specific dependencies for a given requirement. For example, to use a specific branch of the schemas when developing the server hosted by github we can add the line:

```
git+git://github.com/david4096/schemas.git@protobuf31#egg=ga4gh_schemas
```

This informs the installer to resolve dependencies from github before PyPi, allowing the developer to work against a specific version of the schemas under their control.

By explicitly stating a dependency, others can review changes to the data model. When a change has been accepted in the schemas, you can adjust your constraints to point at the current master branch of schemas.

```
git+git://github.com/ga4gh/schemas.git@master#egg=ga4gh_schemas
```

At the time of a release, the same process allows us to specify a precise released version of the schemas and client to develop against.

1.6.3 GitHub workflow

First, go to https://github.com/ga4gh/server and click on the 'Fork' button in the top right-hand corner. This will allow you to create your own private fork of the server project where you can work. See the GitHub documentation for help on forking repositories. Once you have created your own fork on GitHub, you'll need to clone a local copy of this repo. This might look something like:

```
git clone git@github.com:username/server.git
```

We can then install all of the packages that we need for developing the GA4GH reference server:

```
cd server
virtualenv env
source env/bin/activate
pip install -r dev-requirements.txt -c constraints.txt
```

This will take a little time as the libraries that we require are fetched from PyPI and built. You can now start the server using a `python server_dev.py`, or by installing it to the current environment using `python setup.py install` and then running `ga4gh_server`. For more information on using the server, visit *GA4GH API Demo*.

It is also important to set up an upstream remote for your repo so that you can sync up with the changes that other people are making:

```
git remote add upstream https://github.com/ga4gh/server.git
```

All development is done against the `master` branch.

All development should be done in a topic branch. That is, a branch that the developer creates him or herself. These steps will create a topic branch (replace `TOPIC_BRANCH_NAME` appropriately):

```
git fetch --all
git checkout master
git merge --ff-only upstream/master
git checkout -b TOPIC_BRANCH_NAME
```

Topic branch names should include the issue number (if there is a tracked issue this change is addressing) and provide some hint as to what the changes include. For instance, a branch that addresses the (imaginary) tracked issue with issue number #123 to add more widgets to the code might be named `123_more_widgets`.

At this point, you are ready to start adding, editing and deleting files. Stage changes with `git add`. Afterwards, checkpoint your progress by making commits:

```
git commit -m 'Awesome changes'
```

(You can also pass the `--amend` flag to `git commit` if you want to incorporate staged changes into the most recent commit.)

Once you have changes that you want to share with others, push your topic branch to GitHub:

```
git push origin TOPIC_BRANCH_NAME
```

Then create a pull request using the GitHub interface. This pull request should be against the `master` branch (this should happen automatically).

At this point, other developers will weigh in on your changes and will likely suggest modifications before the change can be merged into `master`. When you get around to incorporating these suggestions, it is likely that more commits will have been added to the `master` branch. Since you (almost) always want to be developing off of the latest version of the code, you need to perform a rebase to incorporate the most recent changes from `master` into your branch.

Warning: We recommend against using `git pull`. Use `git fetch` and `git rebase` to update your topic branch against mainline branches instead. See the *Git Workflow Appendix* for elaboration.

```
git fetch --all
git checkout master
git merge --ff-only upstream/master
git checkout TOPIC_BRANCH_NAME
git rebase master
```

At this point, several things could happen. In the best case, the rebase will complete without problems and you can continue developing. In other cases, the rebase will stop midway and report a merge conflict. That is, git has determined that it is impossible for it to determine how to combine the changes from the new commits in the `master` branch and your changes in your topic branch and needs manual intervention to proceed. GitHub has some documentation on how to resolve rebase merge conflicts.

Once you have updated your branch to the point where you think that you want to re-submit the code for other developers to consider, push the branch again, this time using the force flag:

```
git push --force origin TOPIC_BRANCH_NAME
```

If you had tried to push the topic branch without using the force flag, it would have failed. This is because non-force pushes only succeed when you are only adding new commits to the tip of the existing remote branch. When you want to do something other than that, such as insert commits in the middle of the branch history (what `git rebase` does), or modify a commit (what `git commit --amend` does) you need to blow away the remote version of your branch and replace it with the local version. This is exactly what a force push does.

Warning: Never use the force flag to push to the `upstream` repository. Never use the force flag to push to the `master`. Only use the force flag on your repository and on your topic branches. Otherwise you run the risk of screwing up the mainline branches, which will require manual intervention by a senior developer and manual changes by every downstream developer. That is a recoverable situation, but also one that we would rather avoid. (Note: a hint that this has happened is that one of the above listed merge commands that uses the `--ff-only` flag to merge a remote mainline branch into a local mainline branch fails.)

Once your pull request has been merged into `master`, you can close the pull request and delete the remote branch in the GitHub interface. Locally, run this command to delete the topic branch:

```
git branch -D TOPIC_BRANCH_NAME
```

Only the tip of the iceberg of git and GitHub has been covered in this section, and much more can be learned by browsing their documentation. For instance, get help on the `git commit` command by running:

```
git help commit
```

To master git, we recommend reading this free book (save chapter four, which is about git server configuration): Pro Git.

1.6.4 Contributing

See the files `CONTRIBUTING.md` and `STYLE.md` for an overview of the processes for contributing code and the style guidelines that we use.

1.6.5 Development utilities

All of the command line interface utilities have local scripts that simplify development: for example, we can run the local version of the `ga2sam` program by using:

```
python ga2sam_dev.py
```

To run the server locally in development mode, we can use the `server_dev.py` script, e.g.:

```
python server_dev.py
```

will run a server using the default configuration. This default configuration expects a data hierarchy to exist in the `ga4gh-example-data` directory. This default configuration can be changed by providing a (fully qualified) path to a configuration file (see the *Configuration* section for details).

There is also an OpenID Connect (oidc) provider you can run locally for development and testing. It resides in `/oidc-provider` and has a run.sh file that creates a virtualenv, installs the necessary packages, and runs the server. Configuration files can be found in `/oidc-provider/simple_op`:

```
cd oidc-provider
./run.sh
```

The provider expects OIDC redirect URIs to be over HTTPS, so if the ga4gh server is started with OIDC enabled, it defaults to HTTPS. You can run the server against this using:

```
python server_dev.py -c LocalOidConfig
```

For tips on how to profile the performance of the server see ref_server_profiling

1.6.6 Organization

The code for the project is held in the `ga4gh` package, which corresponds to the `ga4gh` directory in the project root. Within this package, the functionality is split between the `client`, `server`, `protocol` and `cli` modules. The `cli` module contains the definitions for the `ga4gh_client` and `ga4gh_server` programs.

1.6.7 Git Workflow Appendix

Don't use `git pull`

We recommend against using `git pull`. The `git pull` command by default combines the `git fetch` and the `git merge` command. If your local branch has diverged from its remote tracking branch, running `git pull` will create a merge commit locally to join the two branches.

In some workflows, this is not an issue. For us, however, it creates a problem in the future. When you are ready to submit your topic branch in a pull request, we ask you to squash your commits (usually down to one commit). Given the complex graph topography created by all of the merges, the order in which git applies commits in the squash is very difficult to reason about and will likely create merge conflicts that you find unnecessary and nonsensical (and therefore, highly aggravating!).

We instead recommend using `git fetch` and `git rebase` to update your local topic branch against a mainline branch. This will create a linear commit history in your topic branch, which will be easy to squash, since the commits are applied in the squash in the order that you made them.

`git pull` does have the `--rebase` option which will do a rebase instead of a merge to incorporate the remote branch. One can also set the `branch.autosetuprebase always` config option to have `git pull` do a rebase by default (i.e. without passing the `--rebase` flag) instead of a merge. This will avoid the issue of squashing a non-linear commit history.

So, in truth, we are really recommending against squashing local branches with many merge commits in them. However, using the default settings for `git pull` is the easiest way to end up in this situation. Therefore, don't use `git pull` unless you know what you are doing.

Squash, then rebase

When updating a local topic branch with changes from a mainline branch, we recommend squashing commits in your topic branch down to one commit before rebasing on top of the mainline branch. The reason for this is that, under the hood, to apply the rebase `git rebase` essentially cherry-picks each commit from your topic branch not in the mainline branch and applies it to the mainline branch. Each one of these applications can cause a merge conflict. It is much better to face the potential of only one merge conflict than N merge conflicts (where N is the number of unique commits in the local branch)!

The difficulty of proceeding the opposite way (rebasing, then squashing) is only compounded because of the unintuitiveness of the N merge conflicts. When presented with a merge conflict, your likely intuition is to put the file in the state that you think it ought to be in, namely the condition it was in after the Nth commit. However, if that state was different than the state that git thinks it should be in – namely, the state of the file at commit X where X<N – then you have only created the potential for more merge conflicts. When the next intermediate commit, Y (where X<Y<N) is applied, it too will create a merge conflict. And so on.

So squash, then rebase, and avoid this whole dilemma. The terms are a bit confusing since both "squashing" and "rebasing" are accomplished via the `git rebase` command. As mentioned above, squash the commits in your topic branch with (assuming you have branched off of the `master` mainline branch):

```
git rebase -i `git merge-base master HEAD`
```

(`git merge-base master HEAD` specifies the most recent commit that both `master` and your topic branch share in common. Normally this is equivalent to the most recent commit of `master`, but that's not guaranteed – for instance, if you have updated your local `master` branch with additional commits from the remote `master` since you created your topic branch which branched off of the local `master`.)

And rebase with (again, assuming `master` as the mainline branch):

```
git rebase master
```

GitHub's broken merge/CI model

GitHub supports continuous integration (CI) via Travis CI. On every pull request, Travis runs a suite of tests to determine if the PR is safe to merge into the mainline branch that it targets. Unfortunately, the way that GitHub's merge model is structured does not guarantee this property. That is, it is possible for a PR to pass the Travis tests but for the mainline branch to fail them after that PR is merged.

How can this happen? Let's illustrate by example: suppose PR A and PR B both branch off of commit M, which is the most recent commit in the mainline branch. A and B both pass CI, so it appears that it is safe to merge them into the mainline branch. However, it is also true that the changes in A and B have never been tested *together* until CI is run on the mainline branch after both have been merged. If PR A and B have incompatible changes, even if both merge cleanly, CI will fail in the mainline branch.

GitHub could solve this issue by not allowing a PR to be merged unless it both passed CI and its branch contained (in addition to the commits it wanted to merge in to mainline) every commit in the mainline branch. That is, no PR could be merged into mainline unless its commits were tested with every commit already in mainline. Right now GitHub does not mandate this strict sequencing of commits, which is why it can never guarantee that the mainline CI will pass, even if all the PR CIs passed.

Developers could also enforce this property manually, but we have determined that not using GitHub's UI merging features and judiciously re-submitting PRs for additional CI would be more effort than fixing a broken test in a mainline branch once in a while.

GitHub has recently introduced Protected Branches, which fixes this issue by mandating a strict sequencing of commits as described above. We have protected all of our trunk branches. The downside of using protected branches is increased developer overhead for each branch: merging PR A targeting trunk branch T immediately makes PR B targeting T out of date and therefore unmergable without pulling in the most recent changes from T and re-running CI on B. However, we think it is worth enabling this feature to prevent broken trunk branches.

Managing long-running branches

Normally, the development process concerns two branches: the feature branch that one is developing in and the trunk branch that one submits a pull request against (usually this is `master`). Sometimes, development of a major feature may require a branch that lives on for a long time before being incorporated into a trunk branch. This branch we call a topic branch.

For developers, the process of submitting code to a topic branch is almost identical to submitting code to a trunk branch. The only difference is that the pull request is made against the topic branch instead of the trunk branch (this is specified in the GitHub pull request UI).

Topic branches do, however, require more management. Each long-lived topic branch will be assigned a branch manager. This person is responsible for keeping the branch reasonably up to date with changes that happen in the trunk branch off of which it is branched. The list of long running branches and their corresponding branch managers can be found here.

It is up to the branch manager how frequently the topic branch pulls in changes from the trunk branch. All topic branches are hosted on the ga4gh/server repository and are GitHub protected branches. That is, there can be no force pushes to the branches, so they must be updated using `git merge` rather than `git rebase`. Updates to topic branches must be done via pull requests (rather than directly on the command line) so that the Travis CI runs and passes prior to merging.

1.6.8 Release process

There are two types of releases: development releases, and stable bugfix releases. Development releases happen as a matter of course while we are working on a given minor version series, and may be either a result of some new features being ready for use or a minor bugfix. Stable bugfix releases occur when mainline development has moved on to another minor version, and a bugfix is required for the currently released version. These two cases are handled in different ways.

Development releases

Version numbers are MAJOR.MINOR.PATCH triples. Minor version increments happen when significant changes are required to the server codebase, which will result in a significant departure from the previously released version, either in code layout or in functionality. During the normal process of development within a minor version series, patch updates are routinely and regularly released.

This entails:

1. Create a PR against `master` with the release notes; presently, the release notes are located in `docs/status.rst`
2. Once this has been merged, tag the release on GitHub (on the releases page) with the appropriate version number.
3. Fetch the tag from the upstream repo, and checkout this tag.

4. Replace git URLs in the `constraints.txt` to point at the schemas and client releases this server release is meant to depend on.
5. Replace git URLs in the `docs/environment.yml` using the same versions as (4).
6. Create the distribution tarball using `python setup.py sdist`, and then upload the resulting tarball to PyPI using `twine upload dist/ga4gh-MAJOR.MINOR.PATCH.tar.gz` (using the correct file name).
7. Verify that the documentation at http://ga4gh-reference-implementation.readthedocs.org/en/stable/ is for the correct version (it may take a few minutes for this to happen after the release has been tagged on GitHub). The release notes docs should have changed, so that is a good section to look at to confirm the change.

Stable bugfix release

When a minor version series has ended because of some significant shift in the server internals, there will be a period when the `master` branch is not in a releasable state. If a bugfix release is required during this period, we create a release using the following process:

1. If it does not already exist, create a release branch called `release-$MAJOR.MINOR` from the tag of the last release.
2. Fix the bug by either cherry picking the relevant commits from `master`, or creating PRs against the `release-$MAJOR.$MINOR` branch if the bug does not apply to `master`.
3. Follow steps 1-6 in the process for *Development releases* above, except using the `release-$MAJOR.$MINOR` branch as the base instead of `master`.

1.7 Status

The GA4GH server is currently under active development, and several features are not yet fully functional. These mostly involve the reads API. Some missing features are:

- Unmapped reads. We do not support searching for unmapped reads.
- Searching over multiple ReadGroups in different ReadGroupSets.

For more detail on individual development issues, please see the project's issue page.

1.7.1 Release Notes

0.3.5

Alpha pre-release supporting minor update. We have done some restructuring, some bug fixes and some minor protocol updates in this release.

- **Restructuring: We have reorganized our codebase to separate out the schemas,** client, and server components as separately installable Python components. Running 'pip install ga4gh –pre' will still automatically install all required packages. But now developers who wish to leverage the Python libraries around the schemas can just use 'pip install ga4gh-schemas', and developers of client code can gain access to the client API library my just running 'pip install ga4gh-client'.
- **Bug fixes and improvements**
  - Fix to be able to handle VCFs with genotype == './.' (server #1389)
  - Fixed bug with OIDC Authentication (server #1452, #1508)
  - Repo manager list out of range error (server #1472)

  - G2P features search was returning a 404, this has been fixed (server #1379)
  - Sped up readGroupSet generator (server #1316)
  - Enhanced ENCODE RNA downloader
  - Added support for Auth0 (server #1452)
  - Refactored usage of the term 'biosample' to be consistent (server #1394)
  - Moved to Protobuf version 3.1 (server #1471)
- **Documentation updates**
  - Github usage
  - Added sections for new Python packages: client and schemas
  - Split out repository manager docs
  - Docker file documentation updated
  - Updated the Configuration section to document the Auth0 settings

0.3.4

Alpha pre-release supporting major feature update.

- **G2P functionality added to support the following API endpoints:**
  - POST */phenotypeassociationsets/search*
  - POST */phenotypes/search*
  - POST */featurephenotypeassociations/search*
- Biometadata tags for RNA quantifications.
- Improvements to the RNA quantification ingestion pipeline.
- Migrated CLI related code to *cli* module.
- Add demonstration RNA quantification data.
- Minor doc fixes

Known Issues

- When searching using a wildcard, *, an Internal Server Error occurs. #1379
- When listing many Read Group Sets, responses can be quite slow causing timeouts. #1316

0.3.3

Alpha pre-release supporting major feature update.

- **RNA functionality added to support the following API endpoints:**
  - POST /rnaquantificationsets/search
  - GET /rnaquantificationsets/{id}
  - POST /rnaquantifications/search
  - GET /rnaquantifications/{id}
  - POST /expressionlevels/search

    - GET /expressionlevels/{id}
- Fixed bug where transcript effects would be repeated within a search result.

0.3.2

Alpha pre-release supporting major feature update.

- Metadata functionality added to support biosample and individual metadata capabilities.
- Now support searching features by 'name' and 'gene_symbol'. These fields have been promoted to facilitate the future integration of the RNA and G2P modules.

0.3.1

Alpha pre-release supporting major feature update. This release is not backwards compatible with previous releases due to several changes to the schemas that were required to move to protocol buffers.

- This release includes the code changes necessary for the migration to protocol buffers from Avro.
- Client applications will need to be rebuilt to the new schemas and use the protobuf json serialization libraries to be compatible with this version of the server.

0.3.0

Alpha pre-release supporting major feature update. This release is not backwards compatible with previous releases, and requires the data files be re-imported.

- File locations are now tracked in a registry.db registry such that the files can be located anywhere. The information from the json sidecar files are also included in the database.
- Ontology terms are now imported via an OBO file instead of the old pre-packaged sequence_ontology.txt file. A sample OBO file has been added to the sample data set for the reference server.
- Added a custom landing page option for the main page of the server.
- Performance improvement for variant search when calls are set to an empty string.
- Improved server configuration including Apache configuration and robots.txt file.

0.2.2

Alpha pre-release supporting major feature update. This release is backwards incompatible with previous releases, and requires a revised data directory layout.

- Added sequence and variant annotations (which introduces a sqlite database component)
- Added repo manager, a command line tool to manage data files and import them into the server's data repository
- Supported searching over multiple ReadGroups, so long as they are all in the same ReadGroupSet and all of the ReadGroups in the ReadGroupSet are specified

0.2.1

Bugfix release that fixes a problem introduced by upstream package changes

0.2.0

Alpha pre-release supporting major schema update. This release is backwards incompatible with previous releases, and requires a revised data directory layout.

- Schema version changed from v0.5 to v0.6.0a1
- Various backwards incompatible changes to the data directory layout
- Almost complete support for the API.
- Numerous code layout changes.

0.1.2

This bugfix release addresses the following issues:

- #455: bugs in reads/search call (pysam calls not sanitized, wrong number of arguments to getReadAlignments)
- #433: bugs in paging code for reads and variants

0.1.1

- Fixes dense variants not being correctly handled by the server (#391)
- Removes unused paths (thus they won't confusingly show up in the HTML display at **/**)

0.1.0

Just bumping the version number to 0.1.0.

0.1.0b1

This is a beta pre-release of the GA4GH reference implementation. It includes

- A fully functional client API;
- A set of client side tools for interacting with any conformant server;
- A partial implementation of the server API, providing supports for variants and reads from native file formats.

0.1.0a2

This is an early alpha release to allow us to test the PyPI package and the README. This is not intended for general use.