NGS 2014 Tutorial - Databases Documentation

Release 0.0

Adina Howe

August 25, 2014

Contents

Contents:

CHAPTER 1

So you want to get some sequencing data in NCBI?

This is a set of tutorials for working with the NCBI and MG-RAST databases – specifically, to download project specific information.

First, let's think about how these databases are structured. I am going to create a database for folks to deposit whole genome sequences. What kind of information am I going to store in this? Many of you may be familiar with such a database, hosted by the NCBI. The scripts that complement this tutorial can be downloaded with the following:

```
git clone https://github.com/adina/scripts-for-ngs2014.git
```

Let's come up with a list of things we'd like stored in this database and discuss some of the challenges involved in database creation, management, and access.

1.1 The challenge

So, you've been given a list of genomes and been asked to create a phylogenetic tree of these genomes. How big would this list be before you thought about hiring an undergraduate to download sequences?

Say the list is only 3 genomes:

```
CP000962
CP000967
CP000975
```

The following are some ways with which I've used to grab genome sequences:

1. Go to the web portal and look up each FASTA
2. Go to the FTP site, find each genome, and download manually
3. Use the NCBI Web Services API to download the data

Among these, I'm going to assume many of you are familiar with the first two. This tutorial then is going to focus on using APIs.

1.2 What is an API and how does it relate to NCBI?

Here's some answers, among which my favorite is "an interface through which you access someone else's code or through which someone else's code accesses yours – in effect the public methods and properties."

The NCBI has a whole toolkit which they call *Entrez Programming Utilities* or *eutils* for short. You can read all about it in the documentation. There are a lot of things you can do to interface with all things NCBI, including publications, etc., but I am going to focus today on downloading sequencing data.

To do this, you're going to be using one tool in *eutils*, called *efetch*. There is a whole chapter devoted to efetch – when I first started doing this kind of work, this documentation always broke my heart. Its easier for me to just show you how to use it.

Open a web browser, and try the following URL to download the nucleotide genome for CB00962:

```
http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nuccore&id=CP000962&rettype=fasta&retmode
```

Note that the NCBI knows a lot about this genome. Check it out here.

If I want to access other kinds of data associated with this genome, I would try the following command:

```
http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nuccore&id=CP000962&rettype=gb&retmode=te
```

Do you notice the difference in these two commands? Let's breakdown the command here:

1. <http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?> This is command telling your computer program (or your browser) to talk to the NCBI API tool efetch.
2. <db=nuccore> This command tells the NCBI API that you'd like it to look in this particular database for some data. Other databases that the NCBI has available can be found here.
3. <id=CP000962> This command tells the NCBI API efetch the ID of the genome you want to find.
4. <rettype=gb&retmode=text> These two commands tells the NCBI how the data is returned. You'll note that in the two examples above this command varied slightly. In the first, we asked for only the FASTA sequence, while in the second, we asked for the Genbank file. Here's some elusive documentation on where to find these "return" objects.

Also, a useful command is also <version=1>. There are different versions of sequences and some times that is useful. For reproducibility, I try to specify versions in my queries, see these comments.

Note: Notice the "&" that comes between each of these little commands, it is necessary and important.

1.3 Automating with an API

Ok, let's think of automating this sort of query.

In the shell, you could run the same commands above with the addition of *curl* on your EC2 instance:

```
curl "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nuccore&id=CP000962&rettype=fasta&r
```

You'll see it fly on to your screen. Don't panic - you can save it to a file and make it more useful.:

```
curl "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nuccore&id=CP000962&rettype=fasta&r
```

You could now imagine writing a program where you made a list of IDs you want to download and put it in a for loop, *curling* each genome and saving it to a file. The following is a script. Thanks to Jordan Fish who gave me the original version of this script before I even knew how and made it easy to use.

To see the documentation for this script:

```
/usr/bin/python fetch-genomes.py
```

You'll see that you need to provide a list of IDs and a directory where you want to save the downloaded files.

To run the script:

```
/usr/bin/python fetch-genomes.py interesting-genomes.txt genbank-files
```

Note: You may want to run this on just a few of these IDs to begin with. You can create a smaller list using the *head* command with the -n parameter in the shell. For example, head -n 3 interesting-genomes.txt > 3genomes.txt.

Let's take a look inside this script. The meat of this script uses the following code:

```
url_template = "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nucleotide&id=%s&rettype=
```

You'll see that the *id* here is a string character which is obtained from list of IDs contained in a separate file. The rest of the script manages where the files are being placed and what they are named. It also prints some output to the screen so you know its running.

1.4 Exercise - Downloading data

Try modifying the fetch_genomes.py script to download just the FASTA sequences of the genes.

Running this script should allow you to download genomes to your heart's content. But how do you grab specific genes from this data then? Specifically, the challenge was to make a phylogenetic tree of sequences, so let's target the conserved bacterial gene, *16S ribosomal RNA gene*.

1.5 Comment on Genbank files

Genbank files have a special structure to them. You can look at it and figure it out for the most part, or read about it in detail here. To find out if your downloaded Genbank files contain 16S rRNA genes, I like to run the following command:

```
grep 16S *gbk
```

This should look somewhat familiar from your shell lesson, but basically we're looking for anylines that contain the character "16S" in any Genbank file we've downloaded. Note that you'll have to run this in the directory where you downloaded these files.

The structure of the Genbank file allows you to identify 16S genes. For example,

```
rRNA        9258..10759
            /gene="rrs"
            /locus_tag="CLK_3816"
            /product="16S ribosomal RNA"
            /db_xref="Pathema:CLK_3816"
```

You could write code to find text like 'rRNA' and '/product="16S ribosomal RNA"', grab the location of the gene, and then go to the FASTA file and grab these sequences. I've done that before.

You could also use existing packages to parse Genbank files. I have the most experience with BioPython. To begin with, let's just use BioPython so you can get to using existing scripts without writing scripts.

First, we'll have to install BioPython on your instance and they've made that pretty easy:

```
apt-get install python-biopython
```

Fan Yang (Iowa State University) and I wrote a script to extract 16S rRNA sequences from Genbank files, here. It basically searches for text strings in the Genbank structure that is appropriate for these particular genes. You can read more about BioPython here and its Genbank parser here.

To run this script on the Genbank file for CP000962:

```
/usr/bin/python parse-genbank.py genbank-files/CP000962.gbk > genbank-files/CP000962.gbk.16S.fa
```

The resulting output file contains all 16S rRNA genes from the given Genbank file.

To run this for multiple files, I use a shell for loop:

```
for x in genbank-files/*; do /usr/bin/python parse-genbank.py $x > $x.16S.fa; done
```

There are multiple ways to get this done – but this is how I like to do it.

And there you have it, you can now pretty much automatically grab 16S rRNA genes from any number of genomes in NCBI databases.

CHAPTER 2

MG-RAST and its API

Just like the NCBI databases, there are many ways you can interact with MG-RAST, and the web interface is possibly the *worst* way.

Another way you could work with MG-RAST is to download the entire database and then write parsers to get what you want out of it. I've also found this incredibly painful but if you want to do that, you can find its database here.

The best way to access MG-RAST data in my experience is to learn to use their API. MG-RAST has done a decent job publishing API documentation – it just takes a bit of practice to understand its structure.

2.1 Example Usage

You read a paper, and the authors reference MG-RAST metagenomes. You want to download these so you can reproduce some of the analysis and ask some of your own questions.

Table S2. Diversity metrics, MG-RAST IDs, and percent of metagenomic reads annotated

Biome type	Sample ID	MG-RAST ID	% of quality reads annotated	Metagenomic richness (S)	Metagenomic diversity (H′)	Bacterial 16S richness (S)	Bacterial 16S diversity (H′)	Bacterial 16S phylogenetic diversity (PD)
Polar desert	EB017	4477900.3	14.5	1,535	6.39	4,527	5.31	300.0
Polar desert	EB019	4477901.3	23.6	1,663	6.52	2,796	3.60	261.1
Polar desert	EB020	4477902.3	17.3	1,376	6.33	4,936	5.79	305.6
Polar desert	EB021	4477903.3	15.9	1,228	6.17	2,845	4.57	195.3
Polar desert	EB024	4477904.3	17.2	1,386	6.34	4,124	5.56	270.0
Polar desert	EB026	4477803.3	20.5	2,231	6.78	2,935	4.92	232.9
Hot desert	MD3	4477805.3	16.4	1,948	6.60	8,895	6.72	485.8
Hot desert	SF2	4477872.3	14.4	1,850	6.56	10,078	6.93	554.4
Hot desert	SV1	4477873.3	17.3	1,981	6.68	9,929	7.14	527.4
Tropical forest	AR3	4477875.3	13.3	1,814	6.51	9,264	5.72	537.1
Boreal forest	BZ1	4477876.3	17.5	2,270	6.79	9,002	6.54	512.9
Temperate deciduous forest	CL1	4477877.3	18.2	2,393	6.81	12,352	7.06	675.0
Temperate coniferous forest	DF1	4477899.3	18.3	2,414	6.81	12,150	6.68	664.6
Temperate grassland	KP1	4477804.3	17.2	2,193	6.72	10,253	6.60	557.4
Tropical forest	PE6	4477807.3	15.6	2,317	6.70	8,772	6.66	476.8
Arctic tundra	TL1	4477874.3	18.8	2,375	6.84	6,965	6.27	437.6

Percentages of quality-filtered shotgun metagenomic reads that could be annotated to functional gene categories and diversity indices calculated from both the shotgun metagenomic data and the 16S rRNA gene amplicon data.

For example, here is some data from a recent PNAS paper, "Cross-biome metagenomic analyses of soil microbial communities and their functional attributes"

If we wanted to download this data with the API, I'd look at the documentation *download*, here. You'll see a couple examples that lists how you would download different stages:

```
http://api.metagenomics.anl.gov/1/download/mgm4447943.3?file=350.1
```

Or...:

```
http://api.metagenomics.anl.gov/1/download/mgm4447943.3?stage=650
```

These two commands above download a specific file or show files from a specific stage for the MG-RAST metagenome ID 4447943.3. You'll notice how they look similar to the NCBI API calls, with a specific structure. You're also requesting specific data with the query terms given after the ID with this *&* structure. Try putting these *urls* into your web browser and you can see the results.

Remember that you can also access the same commands on the shell with the *curl* command, but you need to know what kind of output you expect.

This command outputs a file so you need to save the file to an output:

> curl "http://api.metagenomics.anl.gov/1/download/mgm4447943.3?file=350.1" > 350.1.fastq.gz

This command returns text (in JSON structure):

> curl "http://api.metagenomics.anl.gov/1/download/mgm4447943.3?stage=650"

As a beginner, I often didn't know what to expect and would just try things out – which I recommend as a good way to learn.

Even more useful, I think is the following command:

> http://api.metagenomics.anl.gov/1/download/mgm4447943.3

I like to put this in a web browser because it pretty prints the JSON text output. This command above gives all the dat that can be obtained from the *download* call for this metagenome.

A challenge for MG-RAST is that the types of files and the stages aren't that well-documented. You can get a good guess of what the files and their content from the download page on the web interface, e.g., here. I can tell you from experience that the most important files for me are as follows:

1. File 050.2 - This is the unfiltered metagenome that was originally uploaded to MG-RAST
2. File 350.2 & 350.3 - These are the protein coding genes (amino acids and nucleotides)
3. File 440.1 - These are predicted rRNA sequences (I do not recommend using MG-RAST for sensitive rRNA annotation. It does not use the internal structure of the gene, which other programs appropriately use for classification)
4. File 550.1 - This file shows clustered sequences which are 90% identical, to reduce the number of sequences that need to be annotated. Many folks don't even know that this happens within MG-RAST.
5. File 650.1 & 650.2 - These files are essentially the blat tabular output from comparing your sequence to the database.

A few words on the MG-RAST database. This often confuses people about MG-RAST. The central part of the MG-RAST database is a set of known protein sequences. These known sequences are identified by a unique ID (a mix of numbers and letters). Each known sequence is then related to a known annotation in several databases (e.g., RefSeq, KEGG, SEED, etc.). In other words, the search of your sequences to the database involves a sequence comparison to a sequences in the M5nr sequence database and these sequences are then linked to "a hub" of annotations in several databases. If MG-RAST wants to add another database to its capabilities, it would identify the IDs of sequences related to the sequences in the database. If it existed, the new database annotation would be added to the hub. Otherwise, a new ID would be created and also a new annotation hub. As a consequence of all this, the main thing I work with in MG-RAST is these unique IDs.

2.2 Exercise - Download

Try downloading a few metagenomes from the PNAS paper and associated files. Can you think of how to automate doing this?

MG-RAST annotates sequences and can estimate the abundance of taxonomy and function. Uses structures of databases like SEED, you can thus find broad functional summaries, e.g., the amount of carbon metabolism in various metagenomes.

I work mostly with assemblies, as most of my unassembled are less than 200 bp. In general, I'm also paranoid and like to do any sort of abundance counting on my own. Let me give you an example, if one of my sequences hits two sequences in the MG-RAST database with identical scores, what should one do in the abundance accounting?

2.3 Working with Annotations

Honestly, I'm never sure what MG-RAST is doing, so I like to be in charge of those decisions. Most typically, I am working with 3 types of datasets in any sort of experimental analysis:

1. an annotation file linking my sequence to a database (hopefully one with some structure like SEED),
2. an abundance file (giving estimates of each of my sequences in my database), and
3. some sort of metadata describing my experiment and samples.

MG-RAST can provide you with all three of these, but I typically use it only for #1. This does require a good deal of know-how in scripting land.

To download these annotation files for specific databases (rather than the unique MG-RAST ID), I use the API annotation command. Using the API, I'll select the database I'd like to use and the type of data within that database I would like returned (e.g., function, taxonomy, or unique ID – aka md5sum).

There are a couple examples on the documentation that are worth trying:

```
http://api.metagenomics.anl.gov/1/annotation/sequence/mgm4447943.3?evalue=10&type=organism&source=Sw
```

The above returns a sequence FASTA file with the annotation included in the header of each sequence.:

```
http://api.metagenomics.anl.gov/1/annotation/similarity/mgm4447943.3?identity=80&type=function&source
```

I use this more often. The above returns the BLAT results in a tabular format, including the annotations in the last column. Note that with the *curl* command I can save this to a file and then parse it on my own.

Some comments on the parameters within *type* within these API calls:

1. Organism and function are self-explanatory.
2. Ontology is the "structure" of the database, e.g., Subsystems groups SEED sequences into broader functional groups which have their own unique IDS like SS0001.
3. Feature - This is the most basic ID within the database of choice, e.g., in RefSeq, this would be its accession ID.
4. MD5 - this is the unique ID within MG-RAST.

Note: The other good parameter to be aware of is *version*. This is important to keep all your analysis consistent. And also guarantees that you are working with the most recent database. Also, when you have to go back and repeat the analysis, you'll know what version you used. The problem is that MG-RAST has almost *no* documentation on versions right now. You should write them and complain.

If you do want to download aspects of the database for your analysis, you'll want to explore the documentation for m5nr API calls. With these calls, you can download the various databases you interact with and more importantly, the *ontology* structure of databases.

For example, you can see the information for any md5 ID in RefSeq:

```
http://api.metagenomics.anl.gov//m5nr/md5/000821a2e2f63df1a3873e4b280002a8?source=RefSeq&version=10
```

Or in all MG-RAST databases:

```
http://api.metagenomics.anl.gov//m5nr/md5/000821a2e2f63df1a3873e4b280002a8?version=10
```

If you want to download taxonomy information:

```
http://api.metagenomics.anl.gov/1/m5nr/taxonomy?version=1
```

Or functional information in the SEED:

```
http://api.metagenomics.anl.gov/1/m5nr/ontology?source=Subsystems&min_level=function
```

Note: One of the things you'll notice when you run these commands in the command line with *curl* is that the output is pretty ugly. You'll want to parse these outputs in a programming language you know and look for a JSON parser. I'm most familiar with Python's library json, which can import JSON text into Python libraries easily.

I generally use these downloads to link to my annotations. For example, I'd get the SSID that a sequence might be associated with in a BLAT table download and then link it to the database ontology with a m5nr download call.

2.4 A note on JSON

You might be wondering how to work with these JSON outputs in your own scripting. For example, for this call:

```
curl http://api.metagenomics.anl.gov//m5nr/md5/000821a2e2f63df1a3873e4b280002a8?version=10
```

The output of the raw JSON looks like this:

```
{
   next: "http://api.metagenomics.anl.gov//m5nr/md5/000821a2e2f63df1a3873e4b280002a8?version=10&offset=10",
   prev: null,
   version: "10",
   url: "http://api.metagenomics.anl.gov//m5nr/md5/000821a2e2f63df1a3873e4b280002a8?version=10&offset=0",
 - data: [
    - {
         source: "InterPro",
         function: "Sulfatase",
         type: "protein",
         ncbi_tax_id: 0,
         md5: "000821a2e2f63df1a3873e4b280002a8",
         organism: "Serratia proteamaculans (strain 568)",
         accession: "IPR000917"
      },
    - {
         source: "InterPro",
         function: "Domain of unknown function DUF1705",
         type: "protein",
         ncbi_tax_id: 0,
         md5: "000821a2e2f63df1a3873e4b280002a8",
         organism: "Serratia proteamaculans (strain 568)",
         accession: "IPR012549"
      },
    - {
         source: "InterPro",
         function: "Alkaline phosphatase-like, alpha/beta/alpha",
         type: "protein",
         ncbi_tax_id: 0,
         md5: "000821a2e2f63df1a3873e4b280002a8",
         organism: "Serratia proteamaculans (strain 568)",
         accession: "IPR017849"
      },
```


If you look closely, it looks a lot like a Python *dictionary* structure and that's how most folks interact with it. Since I program mainly in Python, I use its JSON libraries to work with these outputs in my scripting. I installed the library ijson. In your home directory on your instance, install the library:

```
curl https://pypi.python.org/packages/source/i/ijson/ijson-1.1.tar.gz
tar -zxvf ijson-1.1.tar.gz
cd ijson-1.1
python setup.py install
```

You can test that it was installed:

```
python
>>import ijson
>>
```

No error message means you're good to go.

To work with this data structure, I'd look at it first in your pretty JSON-printed webbrowser.

You'll notice that the data is broken down into a set of nested objects. In this example, the first level contains objects like the version, url, and data. If you go into the data object, you'll see nested data about source, function, type, ncbi_tax_id, etc.

I access the specific object "data" in Python with the following code:

```
import urllib
import ijson

url_string = "http://api.metagenomics.anl.gov//m5nr/md5/000821a2e2f63df1a3873e4b280002a8?version=10"

f = urllib.urlopen(url_string)

objects = ijson.items(f, '')
for item in objects:
```

```
    for x in item["data"]:
        print x["function"], x["ncbi_tax_id"], x["organism"], x["source"], x["type"], x["md5"]
```

Now, if I had a much larger object, say the one below, I'd save it to a file first:

```
curl http://api.metagenomics.anl.gov/1/m5nr/taxonomy?version=1 > taxonomy_download.json
```

Then, I would parse through the file:

```
import urllib
import ijson
import sys

f = open(sys.argv[1])
objects = ijson.items(f, '')

for item in objects:
    for x in item["data"]:
        if x.has_key("domain"):
            print x["domain"], x["ncbi_tax_id"]
            #note that not all tax_id's have an associated domain
```

2.5 Exercise - linking MG-RAST to taxonomy

One of the most aggravating searches in MG-RAST is linking a md5sum to its taxonomy. But...once you do it, you can give yourself a huge pat on the back for understanding how to interact with this API.

Can you figure out how to do it? For a given md5sum, identify its taxonomic lineage. What if you had to automate this for several md5sums?

1. Download the BLAT tabular output for mgm4447943.3 (Hint: the file type is 650.2)
2. Identify the best hits for the first 50 reads. (Hint: remember your BLAST tutorial?)
3. Find the taxonomy id associated with the first 50 reads using the API call. (Hint: you're going to want to write your own script for interacting with the following string "http://api.metagenomics.anl.gov//m5nr/md5/" + m5nr + "?source=GenBank")
4. Find the taxonomy lineage associated with that taxon ID (Hint: See this script.

You can also get taxonomy from NCBI returned in XML format:

```
http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=taxonomy&id=376637
```

Another tool I've used is Biopython, which has parsers for XML and Genbank files. Its something I think is worth knowing exists and occasionally I use it, especially for its parsers. Here's a script that I use it for to get taxonomy for a NCBI Accession Number, here and its also in the repo I've been working with during the workshop.

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*