AnnoTree Databases Built Script - Manual

In the following, the bash script "update_annotree_db.sh", which, based on the latest AnnoTree release, automatically builds the protein and mapping databases needed for the application in the DIAMOND+MEGAN pipeline, is described in detail. It takes the download links to the latest versions of the AnnoTree archaea and bacteria databases as command line arguments, utilizes two python scripts and outputs an up-to-date protein sequence database in FASTA format as well as an up-to-date mapping database with mappings of the protein accessions to the NCBI taxonomy, GTDB taxonomy, KEGG orthology IDs, PFAM IDs and TIGRFAM IDs in the SQLite3 DB format.

Prerequisites. In order to run the script, SQLite3 must be installed and a MySQL server needs to be setup and running with a user being created for which the login credentials are known. It is important that no database starting with "gtdb_" is already present in the MySQL server. Furthermore, the MySQL user must have all privileges granted that are needed for creating, editing and deleting MySQL databases.

To run the script, the URLs to the AnnoTree databases are passed as command line arguments like:

```
bash update_annotree_db.sh URL-bacteria URL-archaea
```

The bash script first asks for the MySQL credentials (1, 2), checks them for correctness (4) and, if either the user or the password is incorrect, repeats so until valid credentials are provided (4-9).

```
read -p "MySQL username: " mysqlUsername
read -sp "MySQL password: " mysqlPassword

while ! mysql -u $mysqlUsername -p$mysqlPassword -e"quit"
    echo "MySQL Username or Password incorrect. Please try again."
    read -p "MySQL username: " mysqlUsername
    read -sp "MySQL password: " mysqlPassword
    echo
done
```

When the credentials are verified, a temporary working directory called "annotree_tmp" is created at the path of deployment (10) and the MEGAN specific mapping file for GTDB IDs "gtdb.map" is downloaded from the MEGAN download page into the tmp directory (11).

```
mkdir annotree_tmp
wget -O annotree_tmp/gtdb.map https://.../gtdb.map
```

For each of the two passed URLs (12, 13), the script then downloads (14), extracts (15) and imports (17) the database dump into MySQL while directly removing unneeded data to use as little disk space as possible (16, 19). When successfully imported into MySQL, the database handle is determined in order to access it inside the MySQL server (18).

```
for var in "$@"
do
    wget -O annotree_tmp/annotree.sql.tar.gz "$var"
    tar -xzf annotree_tmp/annotree.sql.tar.gz -C annotree_tmp/
    rm annotree_tmp/annotree.sql.tar.gz
    mysql -u $mysqlUsername -p$mysqlPassword < annotree_tmp/*.sql
    name=$(grep "CREATE DATABASE" annotree_tmp/*.sql -m 1 | grep -o '\bgtdb\w*')
    rm annotree_tmp/*.sql
```

Next, the protein sequence database is built. The relevant data for this is stored in the AnnoTree 'protein_sequences' table. Thus, the script extracts and passes it to a python script ("prot_seqs_to_faa.py") which builds the database in FASTA format and saves it in the tmp directory.

```
    mysql -u $mysqlUsername -p$mysqlPassword -N \
     -e "SELECT * FROM ${name}.protein_sequences;" \
     | python3 scripts/prot_seqs_to_faa.py $name
```

prot_seqs_to_faa.py
The script builds the FASTA protein sequence database but also creates the "accession2gtdb_id.tsv" file that is important for building the mapping database (4, 5). For this, it reads the data from the 'protein_sequences' table line by line (7-11). Each line corresponds to one entry in the table with the attribute data being delimited by tabs. Each line gets stripped to remove the trailing newline command and is split at tabs which yields the gene ID, GTDB ID and protein sequence for that entry (13). Gene ID and GTDB ID get concatenated with a double underscore ("__") to form the accession (14) and possible dots are replaced by underscores (15). This is important to avoid issues with MEGAN which internally processes accessions and splits them at dots. Pairs of accession and GTDB ID are written into the "accession2gtdb_id.tsv" file delimited by a tab (16) and FASTA entries with the accession as the header and the protein sequence are written into the "prot_seqs_&{name}.faa" file (17).

Listing 1: prot_seqs_to_faa.py

```
1  import sys
2  name = sys.argv[1]
3
4  out1 = open("annotree_tmp/accession2gtdb_id.tsv", "w")
5  out2 = open("annotree_tmp/prot_seqs_{}.faa".format(name), "w")
6
7  while True:
8      try:
9          line = input()
10     except EOFError:
11         break
12
13     gene_id, gtdb_id, seq = line.strip().split("\t")
14     accession = "{}__{}".format(gene_id, gtdb_id)
15     accession = accession.replace(".", "_")
16     out1.write("{}\t{}\n".format(accession, gtdb_id))
17     out2.write(">{}\n{}\n".format(accession, seq))
18
19 out1.close()
20 out2.close()
```

At this point, the protein database in FASTA format for the respective AnnoTree database is built and stored in the tmp directory as "prot_seqs_&{name}.faa" and the python script terminates.

Next, the bash script extracts the required data for building the AnnoTree mapping database from the respective source tables and caches it in the tmp directory (21-24). When all relevant data is extracted, the AnnoTree source database can be deleted to free disk space (25).

```
21     mysql -u $mysqlUsername -p$mysqlPassword -N \
        -e "SELECT gene_id, gtdb_id, kegg_id FROM ${name}.kegg_top_hits;" \
        > annotree_tmp/kegg_top_hits.tsv
22     mysql -u $mysqlUsername -p$mysqlPassword -N \
        -e "SELECT gene_id, gtdb_id, pfam_id FROM ${name}.pfam_top_hits;" \
        > annotree_tmp/pfam_top_hits.tsv
23     mysql -u $mysqlUsername -p$mysqlPassword -N \
        -e "SELECT gene_id, gtdb_id, tigrfam_id FROM ${name}.tigrfam_top_hits;" \
        > annotree_tmp/tigrfam_top_hits.tsv
24     mysql -u $mysqlUsername -p$mysqlPassword -N \
        -e "SELECT gtdb_id, ncbi_taxid FROM ${name}.node_tax;" \
        > annotree_tmp/node_tax.tsv
25     mysql -u $mysqlUsername -p$mysqlPassword -e "DROP DATABASE ${name};"
```

Then, another python script ("create_mapping_file.py") is deployed to build the mapping database in accordance to the schema set by the original 'mappings' table.

```
26     python3 scripts/create_mapping_file.py ${name}
```

create_mapping_file.py
The first half of the script reads all the required data from the previously created cache files. It strips and splits each line, extracts relevant information and stores it as key-value pairs in python dictionaries, which are hash tables.
One hash table is created to map the MEGAN specific integers to their corresponding GTDB IDs (4). For this, individual lines of the "gtdb.map" file are read in (5, 6) and after the strip-split operation

(7), the obtained lists are checked if they contain exactly five items (8). This is done because internal nodes of the GTDB taxonomy are also represented in this file which have no genome IDs assigned, thus having one item less per line and by checking the split lines for their length, those without genome ID can be excluded (see Table 1). If the respective list is of length five, the information of interest, namely the custom MEGAN id and the corresponding GTDB can be accessed by index (9, 10) and stored in the hash tables with GTDB ID as key and MEGAN specific integer as value (11).

Listing 2: create_mapping_file.py

```
from random import sample
import sys

cmm_dict = {}   # custom megan mapping dict
with open("annotree_tmp/gtdb.map", "r") as file:
    for line in file:
        line = line.strip().split("\t")
        if len(line) == 5:
            cid = line[0]   # custom megan id
            gid = line[4]   # gtdb id
            cmm_dict[gid] = cid
```

Table 1: Exemplary tabular extract from "gtdb.map". The MEGAN specific integers are stored in the leftmost column and the GTDB genome accessions are stored in the rightmost column.

1000000028	Enterobacteriaceae	-1	5	
1000000067	Escherichia	-1	98	
1000000229	Escherichia flexneri	-1	100	
1950000001	Shigella flexneri	4526576	0	RS_GCF_000953035
1000124975	Escherichia coli O26:H11 10524	5546337	0	RS_GCF_002766295
1000123926	Escherichia coli KCJK6199	5330691	0	RS_GCF_002810665

A second hash table is created to access NCBI taxonomy IDs by their corresponding GTDB IDs (12). The required data is read from the cache file that was created by extracting the information from the AnnoTree 'node_tax' table (13, 14). Each line consists of only the GTDB ID and the NCBI taxonomy ID. However, after stripping and splitting (15), the GTDB ID needs to be split a second time to remove possible version numbers that MEGAN doesn't support (16). Both values then are stored in the hash table with GTDB ID as key and NCBI taxonomy ID as value (17).

```
ntax_dict = {}  # node taxonomy dict
with open("annotree_tmp/node_tax.tsv", "r") as file:
    for line in file:
        gid, tid = line.strip().split("\t") # gtdb id, taxonomy id
        gid = gid.split(".")[0]     # remove version numbers
        ntax_dict[gid] = tid
```

Next, three hash tables are built and filled to access KEGG IDs/PFAM IDs/TIGRFAM IDs by protein accession (18-20). The required data is read from the cache files that were created by extracting the information from the AnnoTree '*_top_hits' tables (23, 24). Since the data is structured identically for all three, a for-loop is used to build each of the three hash tables successively (22). Each line in the files consists of the gene ID, GTDB ID and KEGG/PFAM/TIGRFAM ID and after stripping and splitting (25), gene ID and GTDB ID are concatenated by a double underscore identical to how the accessions were created during the build of the AnnoTree protein database (26, 27). The accession and the corresponding KEGG/PFAM/TIGRFAM ID are then stored in the hash table as key-value pairs, and if the accession already is present in the hash table, the KEGG/PFAM/TIGRFAM ID gets appended to the preexisting list of IDs (28-31).

```
kth_dict = {}       # kegg_top_hits dict
pth_dict = {}       # pfam_top_hits dict
tth_dict = {}       # tigrfam_top_hits dict

for col, dct in [["kegg", kth_dict], ["pfam", pth_dict], ["tigrfam", tth_dict]]:
    with open("annotree_tmp/{}_top_hits.tsv".format(col), "r") as file:
        for line in file:
```

```
            gene_id, gtdb_id, col_id = line.strip().split("\t")
            accession = "{}__{}".format(gene_id, gtdb_id)
            accession = accession.replace(".", "_")
            if accession not in dct:
                dct[accession] = [col_id]
            else:
                dct[accession].append(col_id)
```

When all hash tables are filled, the script opens and reads the "accession2gtdb_id.tsv" file that was created during the protein sequence database built (33, 34). Each line in the file corresponds to one protein sequence and holds its accession together with the associated GTDB ID. The script reads each line and strips and splits it into accession and GTDB ID, based on which the AnnoTree 'mappings' table for the AnnoTree mapping database gets build (35, 36). The GTDB ID is used to obtain the NCBI taxonomy ID (37) and the MEGAN specific GTDB integer (38) from the respective hash tables. Based on the accession, the associated KEGG IDs can be accessed (40). (Note: The functionality for KEGG, PFAM and TIGRFAM is identical here and thus only the code for KEGG is show which is transferable to the other two.) However, due to MEGAN currently only supporting one KEGG ID per accession, if multiple KEGG ID are assigned to one accession, one is randomly picked as the representative (41). Subsequently, the chosen KEGG ID needs to be altered to match MEGAN requirements. For this, the leading letter together with all leading zeros has to be removed (42, 43). If an accession has no corresponding KEGG ID, an empty string is used as a placeholder (45).
Having gathered all the relevant information, the script then formats the data according to the requirements set by the original 'mappings' table and writes them into an output file (49).

```
with open("annotree_tmp/mapping_{}.tsv".format(sys.argv[1]), "w") as out:
    with open("annotree_tmp/accession2gtdb_id.tsv", "r") as file:
        for line in file:
            accession, gid = line.strip().split("\t")
            gid = gid.split(".")[0]   # remove version numbers
            taxonomy = ntax_dict[gid]
            gtdb = cmm_dict[gid]
            try:
                kegg_list = kth_dict[accession]
                kegg = sample(kegg_list, 1)[0]  # random pick
                kegg = kegg[1:]  # drop "K" (K00086 -> 00086)
                kegg = int(kegg)  # drop zeros (00086 -> 86)
            except KeyError:
                kegg = ""
            pfam = ...
            tigrfam = ...

            out.write("{}\t{}\t{}\t{}\t{}\t{}\t{}\t{}\t{}\t{}\t{}\n".format(accession, \
             taxonomy, gtdb, "", "", "", "", kegg, pfam, tigrfam))
```

The mapping files for both the AnnoTree bacteria as well as the AnnoTree archaea databases are stored in the tmp folder and after iterating over the "accessions2gtdb_id.tsv" file the python script terminates.
When the "create_mapping_file.py" script finishes, the files used for caching the AnnoTree data are no longer needed and thus are removed by the bash script(27-31), which ends one iteration of the for-loop (32).

```
    rm annotree_tmp/node_tax.tsv
    rm annotree_tmp/kegg_top_hits.tsv
    rm annotree_tmp/pfam_top_hits.tsv
    rm annotree_tmp/tigrfam_top_hits.tsv
    rm annotree_tmp/accession2gtdb_id.tsv
done
```

When the for-loop finishes after processing both URLs for the AnnoTree bacteria and archaea databases, the bash script removes the "gtdb.map" mapping file (33) and joins and removes the protein sequence FASTA files as well as the two precursor mapping databases (34-37). The resulting "annotree_prot_seqs.faa" file is the finished protein database that DIAMOND can process.

```
rm annotree_tmp/gtdb.map
cat annotree_tmp/prot_seqs_* > annotree_prot_seqs.faa
rm annotree_tmp/prot_seqs_*
cat annotree_tmp/mapping_* > annotree_tmp/mapping.tsv
rm annotree_tmp/mapping_*
```

Subsequently, the AnnoTree mapping database is built by importing the 'info' table, which was copied from the original mapping database and extended to include PFAM and TIGRFAM, into a new SQLite3 database called "annotree.db" (38) and creating and filling the 'mappings' table in the same database (39-43).

```
38 cat scripts/sqlite_info.sql | sqlite3 annotree.db
39 sqlite3 annotree.db <<'END_SQL'
40 CREATE TABLE mappings (Accession PRIMARY KEY , Taxonomy INT, GTDB INT, EGGNOG INT, \
       INTERPRO2GO INT, SEED INT, EC INT, KEGG INTEGER, PFAM INTEGER, TIGRFAM INTEGER) \
       WITHOUT ROWID;
41 .separator \t
42 .import annotree_tmp/mapping.tsv mappings
43 END_SQL
```

The resulting "annotree.db" file is the finished mapping database that Meganizer can process.
After removing the now obsolete mapping file and tmp direcory (44, 45), the bash script terminates and the desired protein and mapping databases are located in the same directory as the bash script.

```
44 rm annotree_tmp/mapping.tsv
45 rm -r annotree_tmp
```