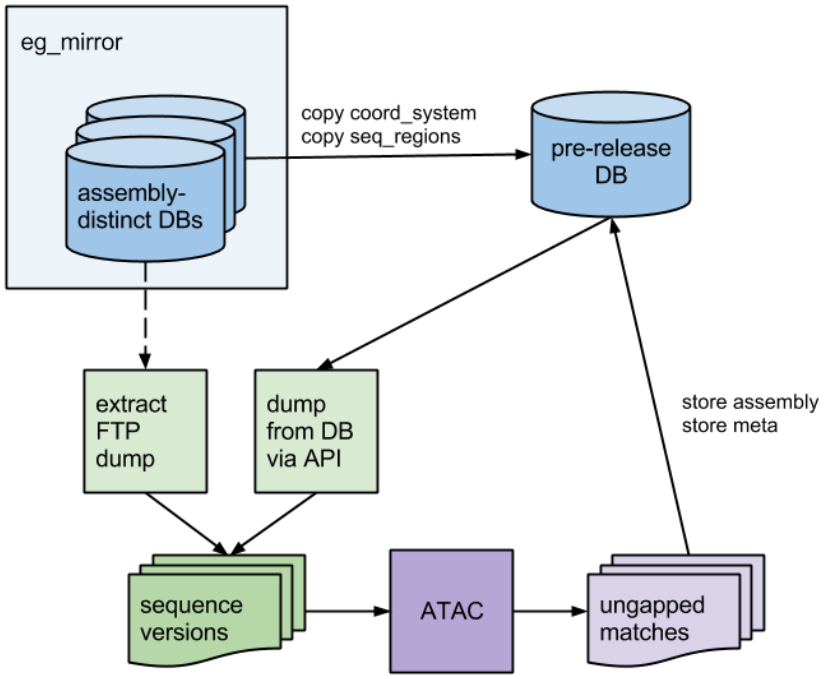
Assembly Mapping (ATAC)

AtacAssemblyConverter can load ATAC assembly-to-assembly matches into the Ensembl schema. It does so by reading from and writing to databases directly, using a very lightweight python ORM.

The process is roughly as follows:

- list databases with distinct assembly versions across a species
- for all old versions:
    - fetch toplevel seq_regions and required coord_systems
    - save the above in current database
    - acquire dna sequence
    - create ATAC mapping to current version
    - save mapping in assembly table
    - save metadata mapping information



Setup

This module is written in python and requires SQLAlchemy (ORM), and SQLSoup (self configuring interface to SQLAlchemy).

A python virtual environment is available in the Ensembl software environment.

```
bsub -q production-rh7 -R "rusage[mem=16000]" -M 16000 -Is bash

# If you already source the environment in your .bashrc, you won't need the following line
source /nfs/software/ensembl/latest/envs/basic.sh

eval "$(pyenv init -)"
pyenv shell atac_assembly_mapping
```

Now fetch code from Git and run it:

```
git clone git@github.com:EnsemblGenomes/eg-assemblyconverter.git
cd eg-assemblyconverter
python atac_ensembl.py -h
```

Note that the sequence dumping script requires the Ensembl API (see Your Perl Version for details) which is usually as simple as just:

```
source /nfs/panda/ensemblgenomes/apis/ensembl/current/setup.sh
```

Usage examples

```
# list all species in Plants
# (divisions names are listed in the help output 'show-species -h')
python atac_ensembl.py show-species EnsemblPlants
['aegilops_tauschii', 'arabidopsis_lyrata', 'arabidopsis_thaliana', 'brachypodium_distachyon', ...

# let's select Oryza glaberrima from this list
# which databases have distinct assembly versions on eg-mirror?
python atac_ensembl.py show-dbs oryza_glaberrima
['oryza_glaberrima_core_10_63_51', 'oryza_glaberrima_core_19_72_2']

# now let's create mappings from all versions on mirror to a non-released version (20_73_2) on eg-devel-1
# (db names are listed in the help output 'map-current -h')
python atac_ensembl.py map-current eg_devel1 oryza_glaberrima_core_20_73_2

# if you have no write permission in the hard-coded locations you can choose your own writable dirs
python atac_ensembl.py map-current plants_prod1 hordeum_vulgare_core_43_96_3 --fasta_dir $ensmapatac --atac_dir
$ensmapatac

# or create mappings from all old versions (10_63_51) to the most current one (19_72_2) on mirror
# and store them on eg-devel-1
python atac_ensembl.py map-historic eg_devel1 oryza_glaberrima

# Lets map between two arbitrary databases (i.e. between species)
# (See "python atac_ensembl.py map-explicit -h")
python atac_ensembl.py map-explicit eg_prod3 \
  triticum_urartu_core_26_79_1 aegilops_tauschii_core_26_79_1

# Note that the above command will populate the coord_system, seq_region, and assembly tables
# which may not be desirable. If you don't want this, just use the mapping report, below.
```

Chain files

See Projecting genes and variants mapped to previous assembly to learn how to produce chain files from your mappings.

Mapping report

This simply runs ATAC and produced an HTML report on the coverage. It will not load the results into the assembly table (unlike atac_ensembl.py). This will work between any two databases.

Note that the examples above will also create an HTML mapping report (block lengths, coverage, contingency table if small enough) in the same directory as the ATAC mapping file. Look at the output for its path. What?

If you're interested in just the mapping report, then run the following:

```
python atac_report.py -h # show help and server names
python atac_report.py --server eg_staging1 --fromDB oryza_barthii_core_25_78_3 --toDB
oryza_glaberrima_core_25_78_2
```

ATAC coverage

Coverage is the percentage of source sequence that can be mapped 1:1 to the target sequence using the ATAC output. It measures how many basepairs (and possibly features living on these basepairs) are lost in the mapping process. * Repeat content can have a large impact on these figures.

species	from release	to release	% mapped
oryza indica	59_1	73_2	95.91
oryza glaberrima	63_51	73_2	93.62
arabidopsis thaliana	59_9	73_10	99.99
vitis vinifera	56_1	73_3	92.74
brachypodium distachyon	56_10	73_12	99.59
vitis vinifera	56_1	73_3	92.74
populus trichocarpa	56_11	73_20	70.77
triticum urartu	ASM34733v1	Aet_v4.0	58.13
hordeum_vulgare	88_2	96_3	24.31*

ATAC mapping using standalone map_explicit.py

to deal with lack of pipeline command line option map_explicit functionality was copy-pasted 👒 into the map_explicit.py.

It can use different source and destination servers and options (--host, --port, –user, --pass and --from_host, --from_port, --from_user, --from_pass) allow to use custom server for running atac projections. The scripts tries to locate *dump_toplevel.pl* sctipt in the same directory it's placed to, but this can be overriden with the oprtion:

```
  --toplevel_dumper  ensembl.prod/eg-assemblyconverter/dump_toplevel.pl
```

--from_{host,port,user,pass} parameters can be ommited and are equal to the corresponfing --{host,port,user,pass} ones.

The script can be used like this

map_explicit.py usage

```
python ensembl.prod/eg-assemblyconverter/map_explicit.py \
>   $(db-to-w details script) \
>   $(db-from details prefix_from_) \
>   --from_db anopheles_funestus_core_1904_95_1 \
>   --to_db anopheles_funestus_core_1906_95_3 \
>   --output_dir AfunF1_AfunF3
```

ie. with bsub:

bsub to map_explicit

```
OUT_DIR=<path_to_store_atac_and_tmp_stuff>
ENS_DIR=<path_to_the_ENSEMBL_ROOT_DIR>
FROM_CMD=<from_sql_alias>
FROM_DBNAME=<source_database_name>
TO_CMD=<to_sql_alias_writable>
DBNAME=<target_database_name>

echo "projecting toplevel cs by atac from ${FROM_CMD}:${FROM_DBNAME} to ${CMD}:${DBNAME}" > /dev/stderr

mkdir -p "$OUT_DIR"

MAP_SCRIPT=$ENS_DIR/eg-assemblyconverter/map_explicit.py
MAP_CMD="python $MAP_SCRIPT \
 $($FROM_CMD details prefix_from_) --from_db $FROM_DBNAME \
 $($CMD details script) --to_db $DBNAME \
 --output_dir $OUT_DIR"

BSUB_OPTS="-q production-rh7 -R 'rusage[mem=16000]' -M 16000"

bsub $BSUB_OPTS \
  -Is "source $ENS_DIR/setup.sh; eval \"\$(pyenv init -)\"; pyenv shell atac_assembly_mapping; $MAP_CMD; exit"
```

Debugging

Some useful SQL to look at your assembly table with:

```
# See tables being updated:
SELECT
  TABLE_NAME, TABLE_TYPE, TABLE_ROWS, TABLE_COMMENT, CREATE_TIME, UPDATE_TIME
FROM
  information_schema.TABLES
WHERE
  TABLE_SCHEMA = DATABASE()
ORDER BY
  UPDATE_TIME DESC
LIMIT 10;

# Loading this into memory simplifies the following query
SELECT * FROM coord_system;

SELECT
  asm.coord_system_id,
  cmp.coord_system_id,
  COUNT(*)
FROM
  assembly
INNER JOIN
  seq_region asm ON asm_seq_region_id = asm.seq_region_id
INNER JOIN
  seq_region cmp ON cmp_seq_region_id = cmp.seq_region_id
GROUP BY
  asm.coord_system_id,
  cmp.coord_system_id;
```

Before the wonders of the Ensembl software environment, it was necessary to create a virtual python environment; the snippet below shows how this was done. It should no longer be necessary, but is left here for reference.

Create virtual environment

```
bsub -q production-rh7 -R "rusage[mem=4000]" -M 4000 -Is bash
cd /nfs/panda/ensemblgenomes/development/grabmuel/ensembl
curl -O https://pypi.python.org/packages/source/v/virtualenv/virtualenv-X.X.tar.gz

# OR
wget https://github.com/pypa/virtualenv/archive/15.1.0.tar.gz

tar xvfz virtualenv-X.X.tar.gz
cd virtualenv-X.X
python virtualenv.py atacVE --system-site-packages
source atacVE/bin/activate
pip install sqlalchemy sqlsoup argparse html
```