CLdb (CRISPR Loci Database) tutorial

last updated: 8/15/13

Preparing genbank files for compatibility with ITEP

Reasoning

ITEP will add PEG IDs to each CDS feature in a genbank unless the PEGs are already provide (by SEED for example).

The genbank files do not need PEG IDs for CLdb, but it can be helpful for analyses of CRISPR associated genes, especially for when location information is involved.

Pipeline

alpha-numeric ordering of scaffolds

```
genbank_contig_reorder.pl < file.gbk > file.order.gbk
```

adding ITEP PEGs

```
addItepIdsToGenbank.py -t file.order.gbk raw.txt file.order.ID.gbk
```

merging files for analyses requiring a closed genome

```
union -sequence file.order.ID.gbk  -outseq file_merged.gbk -osformat2 gb -feature
```

Example database setup

Files required

- Loci table (tab-delimited); columns need:
  - Taxon_ID
  - Taxon_Name
  - Subtype
  - Locus_Start
  - Locus_End

- Operon_Start*
- Operon_End*
- CRISPR_Array_Start*
- CRISPR_Array_End*
- Array_status**
- Operon_status***
- Genbank
- Array_File
- Author
- File_Creation_Date

"*" blank values allowed

"**" Possible values: "present", "absent"

"***" Possible values: "intact", "absent", "broken", "shuffled"

```
"broken" = some genes missing
"shuffled" = gene order
```

- Array table files (tab-delimited; copy and paste from CRISPRFinder); columns needed:
  - Start position
  - Direct repeat sequence
  - Spacer sequence
  - End position
- Genbank files for each organism of interest
  - merged
  - FIG-PEG IDs for CDS features in db_xref tags (e.g. "fig|6666666.40253.peg.2362")

Directory setup

```
The directory name for this example: './CLdb/'
The example loci table: 'loci.txt'

$ mkdir CLdb
$ cd CLdb
$ mkdir genbank
```

- place/symlink genbank files in this directory
  $ mkdir array
- place/symlink array files in this directory

Initial DB construction

making the tables in the database

```
$ CLdb_makeDB.pl -r
```

loading the loci table

```
$ CLdb_loadLoci.pl -d CLdb.sqlite < loci.txt
```

adding number of scaffolds to the loci table

```
$ CLdb_addScaffolds.pl -d CLdb.sqlite
```

Spacers and direct repeats

loading arrays and direct repeats to their respective tables

```
$ CLdb_loadArrays.pl -d CLdb.sqlite
```

grouping spacers and direct repeats (groups with same sequence)

```
$ CLdb_groupArrayElements.pl -d CLdb.sqlite -s -r
```

pseudo-hierarchical clustering of spacers & DRs (good for plotting loci)

```
$ CLdb_hclusterArrays.pl -d CLdb.sqlite -s -r
```

calculating direct repeat consensus sequences

```
$ CLdb_loadDRConsensus.pl -d CLdb.sqlite
```

pairwise blast of all spacers

```
$ CLdb_spacerPairwiseBlast.pl -d CLdb.sqlite
```

- used for plotting & checking for paritial overlap of spacers

---

CRISPR-associated genes

getting genes in CRISPR locus region (defined in Loci table)

```
$ CLdb_getGenesInLoci.pl -d CLdb.sqlite > gene_table.txt
```

- manually currate the 'gene_alias' column values

loading genes into the Genes table

```
$ CLdb_loadGenes.pl -d CLdb.sqlite < gene_table.txt
```

Leader region

getting potential leader regions

```
CLdb_getLeaderRegions.pl -d CLdb.sqlite > possible_leaders.fna
```

getting potential leader regions for just 1 subtype

```
CLdb_getLeaderRegions.pl -d CLdb.sqlite -q "AND subtype='I-B'" > leaders_IB.fna
```

identifying leaders

```
mafft --adjustdirection leaders_IB.fna > leaders_IB_aln.fna
```

- if 2 leaders written for a locus, remove the 1 that does not align
- determine where leader conservation ends
  - for example: conservation ends 50bp from end of alignment
  - this will be trimmed off of the leader region when added to the database

loading identified leader regions

```
CLdb_loadLeaders.pl -d CLdb.sqlite -t 50 test_leader_Ib.fna test_leader_Ib_aln.fna
```

- '-t 50' = trim off the last 50bp of unconserved sequence in the alignment
  - 50bp trimmed from side farthest from the array
- both the aligned and unaligned sequenced are needed because mafft can alter orientation during alignment (–adjustdirect)

grouping leaders (100% sequence identity)

```
CLdb_groupLeaders.pl -da CLdb.sqlite
```

Workflows

Get a fasta

Getting a fasta of all spacers

```
$ CLdb_array2fasta.pl -d CLdb.sqlite > spacers.fna
```

Getting a fasta of all spacers for a particular subtype

```
$ CLdb_array2fasta.pl -d CLdb.sqlite -sub "I-B" > spacers_IB.fna
```

Getting a fasta of all spacers for 2 subtypes

```
$ CLdb_array2fasta.pl -d CLdb.sqlite -sub "I-B" "I-C" > spacers_IB_IC.fna
```

Getting a fasta of all direct repeats

```
$ CLdb_array2fasta.pl -d CLdb.sqlite -r > DR.fna
```

Getting a fasta of all direct repeat consensus sequences

```
$ CLdb_DBconsensus2fasta.pl -d CLdb.sqlite > DR_consensus.fna
```

Spacer BLASTs vs subject genomes or other databases

```
$ CLdb_spacerBlast.pl -d CLdb.sqlite -subject A_woodii.fna 931626.1 "Acetobacterium woodii"
```

- "A_woodii.fna" is the fasta of the genome that will be BLASTed against
- "931626.1" is the taxon_id (FIG_ID)
- "Acetobacterium woodii" is the taxon_name
- blastn-short is used to BLAST selected spacer groups against the genome
- the blast results are then stored in CLdb

Getting information for spacer/DR groups IDs (Example: from table of BLAST hits)

```
$ CLdb_arrayGroup2Info.pl -d CLdb.sqlite < spacer_groups_blastn.txt > array_info.txt
```

Get PAMs (from spacer BLASTs) for 1 CRISPR subtype

```
$ CLdb_getPAMs.pl -da CLdb.sqlite -subtype I-B
```

- Spacer vs subject BLASTs must be performed first

Get spacer-spacer blast hits

All spacer pairwise blast hits

```
$ CLdb_getSpacerPairwiseBlast.pl -d CLdb.sqlite
```

All spacer pairwise blast hits that only partially overlap (possible multiple aquisitions)

```
$ CLdb_getSpacerPairwiseBlast.pl -d CLdb.sqlite -o 0 0.99
```

Get GFF3 of spacer blast hits

GFF3 of all spacers that hit E.coli

```
$ CLdb_spacerBlast2GFF.pl -d CLdb.sqlite -staxon_name "E.coli" > ecoli_hits.gff
```

GFF3 of all spacers that hit FIG|2209.27

```
$ CLdb_spacerBlast2GFF.pl -d CLdb.sqlite -staxon_id 2209.27 > 2209.27_hits.gff
```

Classifying arrays based on direct repeats

Classifying all 'rogue' (no operon) arrays

```
$ CLdb_classifyArraysByDR.pl -d CLdb.sqlite
```

Get idea of classification accuracy

```
$ CLdb_classifyArraysByDR_validate.pl -da CLdb.sqlite
```