# 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_addScaffoldCount.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_DRconsensus2fasta.pl -d CLdb.sqlite > DR_consensus.fna
```

---

### Spacer BLASTs vs subject genomes or other databases

#### BLASTn against 1 subject genome

```
$ 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 (the subject)
- "931626.1" is the subject's taxon_id (FIG_ID)
- "Acetobacterium woodii" is the subject's taxon_name
- blastn-short is used to BLAST selected spacer groups against the genome
- the blast results are then stored in CLdb

#### BLASTn against multiple genomes

```
Make a 3-column, tab-delimited table: "fasta    subject_taxon_id    subject_taxon_name"

This is the 'subject_list' table that will be used for BLASTing against all subjects.

$ CLdb_spacerBlast.pl -d CLdb.sqlite -subject subject_list.txt
```

---

### 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
```

---

## Plotting loci

Multiple elements can be used to create a loci plot from CLdb. These include:

- A 'dna_segs' table (required). This includes start-stop info for all CRISRP and gene features.
- An 'xlims' table (required). This provides the start-stop of the loci to plot.
- A tree showing loci relatedness. This orders the loci by the tree.
- A 'comparisons' table. This shows comparisons (e.g. BLASTp) between spacers or genes from adjacent loci in the plot.

The plotting script names have the table name involved first (e.g CLdb_dna_segs_make.pl), so hopefully they are easier to find by tab-completion

### Making a dna_segs table

- Just selecting Subtype I-A in this example:

  $ CLdb_dna_segs_make.pl -d CLdb.sqlite -sub I-A > dna_segs_I-A.txt

- Also get gene cluster information from an ITEP database:

  $ CLdb_dna_segs_make.pl -d CLdb.sqlite -sub I-A -I DATABASE.sqlite all_I_2.0_c_0.4_m_maxbit > dna_segs_I-A.txt

- Just completely intact loci (no 'broken')

  $ CLdb_dna_segs_make.pl -da CLdb.sqlite -sub I-A -q "AND loci.operon_status != 'broken'" > dna_segs_I-A.txt

### Making an xlims table

- Just subtype I-A in this example:

  $ CLdb_xlims_make.pl -da CLdb.sqlite -sub I-A > xlims_I-A.txt

- Just completely intact loci (no 'broken')

  $ CLdb_xlims.pl -da CLdb.sqlite -sub I-A -q "AND loci.operon_status != 'broken'" > xlims_I-A.txt

**Ordering the dna_segs table by a tree**

- Needed if a tree is added to the plot. The tree is pruned to just the taxa in the dna_segs table. Leaves will be added if any taxa have multiple loci in the dna_segs table.

  $ CLdb_dna_segs_orderByTree.pl -t tree.nwk < dna_segs.txt > dna_segs_order.txt

**Ordering the xlims table by a tree**

- Needed if a tree is added to the plot. The same tree editting will be done as with a dna_segs table, but an editted tree will not be written.

  $ CLdb_dna_segs_orderByTree.pl -t tree.nwk < xlims.txt > xlims_order.txt

**Making a comparisons table**

- Use an tree-ordered dna_segs table if plotting with a tree

  $ CLdb_comparison_make.pl -da CLdb.sqlite < dna_segs_order.txt > comparisons.txt

**Formatting colors of genes, spacers, and direct repeats**

- This will make the feature colors more discernable and remove any coloring that is not needed to descriminate related features in different loci (i.e. coloring is not needed if a related feature is only found in adjacent loci and comparisons are connecting them all).

  $ CLdb_dna_segs_formatColor.pl -c comparisons.txt < dna_segs_order.txt > dna_segs_order_col.txt

**Plotting with R functions**

- A set of functions can be used to pull in and do some final edits on all of the plot elements (tables & tree). These functions (along with an example at the end of the script) are found in:

  $ CLdb_loci_plot_func.r