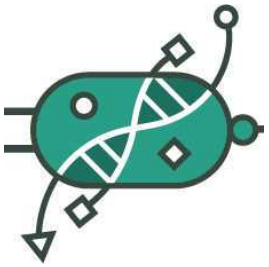
Oct 20, 2025 Version 4

Bacterial genome annotation script using BLASTN V.4



DOI
https://dx.doi.org/10.17504/protocols.io.dm6gpjrb1gzp/v4

Lewis Grozinger[1], Ana Mariya Anhel[2], Lorea Alejaldre[1,2], Ángel oñi-Moreno[1,2]

[1]Centro Nacional de Biotecnologia (CNB-CSIC) Madrid, Spain;
[2]Centro de Biotecnología y Genómica de Plantas, Universidad Politécnica de Madrid (UPM)-Instituto Nacional de Investigación y Tecnología Agraria y Alimentaria (INIA/CSIC), Madrid, Spain

Ángel oñi-Moreno: angel.goni@upm.es;

Biocomputation Lab
CNB-CSIC

OPEN ACCESS

DOI: https://dx.doi.org/10.17504/protocols.io.dm6gpjrb1gzp/v4

Protocol Citation: Lewis Grozinger, Ana Mariya Anhel, Lorea Alejaldre, Ángel oñi-Moreno 2025. Bacterial genome annotation script using BLASTN. **protocols.io** **https://dx.doi.org/10.17504/protocols.io.dm6gpjrb1gzp/v4**Version created by **Lewis Grozinger**

Protocol status: Working
We use this protocol and it's working

Created: June 19, 2025

Last Modified: October 20, 2025

Protocol Integer ID: 220590

Keywords: Genome anotation, Bacterial, P. putida, Transposon, Transposon library, E. coli, bacterial genome annotation script, bacterial genome, genome of pseudomonas putida kt2440, transposon integration events in genome, genome alignment, annotations of the transposon sequence, reads to transposon sequence, method of genome alignment, alignment to the genome, annotations from genbank, positions in the genome, other genome, sequencing read, position in the genome, genome, gene locus of transposon insert, genome sequence, multiple genome, genbank files for each read, multiple genome sequence, blastn this protocol, genbank format, pseudomonas putida kt2440, genbank file, genbank sequence, multiple genbank file, transposon sequence, using blastn, transposon insert, genbank reference, gene, python package tnatla, sequence trimming, genbank, gene locus, transposon integration event, new tool for metadata annotation, general metadata annotation of the result, reads with corresponding feature, blastn, general metadata annotation,

Funders Acknowledgements:
Comunidad de Madrid
Grant ID: Y2020/TCS-6555,2019-T1/BIO-14053
MCIN/AEI
Grant ID: CEX2020-000999-S,PID2020-117205GA-I00,CNS2022-135951, PID2023-152470NB-I00
ERC
Grant ID: 101044360

Abstract

This protocol uses the command line tools provided by the Python package TnAtlas to identify and annotate transposon integration events in genomes.

Given a set of sequencing reads, transposon sequences and genomes, the protocol looks for positions in the genomes where the transposon sequences have been integrated, annotates the sequencing reads with corresponding features of the genome, and presents a summary of the results as a spreadsheet. Integrations are identified by aligning reads to transposon sequences using NCBI's BLASTN, and annotations are made from reference genomes in GENBANK format.

We have used this protocol in our group (https://biocomputationlab.com) to identify the location and gene locus of transposon inserts in the genome of Pseudomonas putida KT2440. However, this script can be used for other genomes for which the genome sequence and annotation are available.

This updated version of the protocol is the first to be supported by the Python package TnAtlas, which contains library code which can be used to run the protocol programmatically (in a Python script or notebook), as well as compute and present other statistics about the results themselves.

The package TnAtlas requires Python versions greater than or equal to 3.8. The protocol requires at least blastn version greater than or equal to 2.12. Optionally, the protocol also requires sickle version 1.33 or greater, and fastqc (any version), in order to perform the sequence trimming and quality control reporting.

This is a description of the LAP entry LAPu-InsertsGenAnnotation-2.0.0 located in the **LAP repository**, specifically **LAPu-InsertsGenAnnotation-3.0.0** and **Github Entry TnAtlas**, 2 places that you can download directly the tool used and usage examples.

The major changes from previous version are:

1. Now shipped as a Python package: The code and tools are packaged and can be installed with Python PIP.
2. New tool for metadata annotation: The old --summary-map system has been replaced with a dedicated tool `tnmeta`, for general metadata annotation of the results.
3. Command line options for saving intermediate files: By default, the tools create far fewer files. The options, --sam, --trim-save, --transposon-save, and --genome-save control the creation of intermediate files.
4. Method of genome alignment: Alignment to the genome is now based on aligning the reads after the transposon sequence that has been found has been removed, giving preference to alignments earlier in the read, and also avoiding position errors of a few base pairs that can affect sequence logo results.
5. Generates annotated genbank sequences: tnfind generates genbank files for each read that includes annotations of the transposon sequence and of the corresponding features from its position in the genome.
6. Annotations from genbank: Annotations come from genbank reference genomes instead of separate CSV files.
7. Multiple genomes: `tnfind` can search through multiple genome sequences at once by passing either passing multiple records in a genbank file, or multiple genbank files, to the `-genome` option.

Materials

Software

- Docker or Docker Desktop

OR

- Linux or MacOS
- Python 3.8 or greater
- BLAST+ version 2.12 or greater
- Sickle
- FastQC

Safety warnings

This protocol works with Linux and MacOS and Windows systems, but the quality control and sequence trimming features require the installation of Docker if using Windows.

Prepare input data

1 **Reference genome**

You can download the genome of the organism(s) that you want to compare to your sequencing reads from different sources such as NCBI, GSA or even pages dedicated to the organism (for example **pseudomonas.com**).

For this protocol, the genome files must be in **GENBANK, FASTA, or FASTQ** format.

Note

Annotation will only work if GENBANK files are provided containing sequence features.

2 **Sequencing reads**

Place all read files under a single directory. These files can be in either FASTA or FASTQ format, but formats cannot be mixed (i.e. the tool will not process both FASTA and FASTQ files simultaneously).

Note

The file extension is largely irrelevant and can be specified later using the --input-ext option.

Note

For quality control and read trimming the FASTQ format is required. Sanger, Solexa/Illumina1.0 and Ilumina>1.3 quality score encodings are supported for FASTQ files.

Each FASTA or FASTQ file can contain multiple reads, and multiple files can be provided. All reads within all FASTA or FASTQ file in the directory will be processed. It is therefore important that each read has a unique ID (the ID of the sequence follows the first > character in the FASTA format, and the first @ character in FASTQ format).

Note

Later, when attaching (optional) metadata to the results, the ID will also be used to identify the plate and well from which the read was sampled. It is therefore recommended that IDs are generated in a systematic way which facilitates this identification.

3 **Transposon sequences**

Sequences belonging to the transposon systems used should be provided in either GENBANK, FASTA, or FASTQ format. These sequences will be used to determine whether, and from where, to start aligning a read to the reference genomes.

Each GENBANK, FASTA, or FASTA file can contain multiple sequences, and multiple files can be provided.

Note

Shorter sequences tend to give better results and less false positives than longer ones. But of course one should avoid sequences that are extremely short. Sequences should certainly be longer than the blastn word size (default: 11 base pairs).

4 **Sample plate metadata**

Additional metadata can be attached to individual sequencing reads by providing the metadata in a grid, where position in the grid can be mapped onto samples in a well plate.

The grid, or grids, can be provided in excel format (XLSX). The first row of the excel file should contain numbers that index the columns of the plate, and the first column should contain letters that index the rows of the plate. Each cell corresponding to a well in a plate should contain the metadata that you wish to attach to the sequencing reads originating from that well.

The name of the files themselves should follow the format:

<PLATE_IDENTIFIER>_<METADATA_LABEL>.xlsx

where <PLATE_IDENTIFIER> is a unique identifier for the plate to which the metadata should be applied, and <METADATA_LABEL> is the name that you would like for the column containing the metadata that will appear in the output table.

<PLATE_IDENTIFIER> should usually correspond to some unique part of the IDs of the sequencing reads (see the note on IDs in go to step #2

An example metadata file is provided here:
22CCRAA01_identity.xlsx 18.7KB

Which will attach a metadata column called "identity" to the reads that come from samples in the "22CCRAA01" plate.

5 If you have a **Windows** system, follow the instructions for **"Windows systems"**

If you have a **MacOS** system you can follow either **"MACOS systems"** or **"Docker on MacOS"**

If you have a **Linux** system you can follow either **"Linux systems"** or **"Docker on Linux"**

Note

Using Docker will mean having to install less dependencies manually, you will just have to install and setup Docker.

STEP CASE

Linux systems 25 steps

6 **Install Python3**

Note

If you already have Python3 installed and setup you may use any Python greater than or equal to version 3.8.

Most Linux distributions have Python3 in their package repositories, and it is convenient to install Python3 using the package manager. This protocol requires a Python version of at least 3.8.

Command

Install Python3 using apt package manager

```
sudo apt-get update
sudo apt-get install python3
```

Note

The latest, up-to-date instructions for installing Python on Linux are **here**.

It is also important to have `pip` installed. If pip does not come with your Python3 installation, you must install it manually.

Command

Install pip

```
python3 -m ensurepip --upgrade
```

Note

The latest, up-to-date instructions for installing pip on Linux are **here**.

7 **Install BLAST+**

There are different ways to install the NCBI-BLAST+ suite of tools. Fortunately most Linux distributions provide a package called **ncbi-blast+** or similar that can be installed

with the package manager. If your distribution has such a package in its repositories, this is the recommended way to install BLAST+.

Command

Install NCBI BLAST+ suite using apt package manager

```
sudo apt-get install ncbi-blast+
```

If no BLAST package is available for your distribution, then the latest versions of the tools can also be downloaded from the NCBI ftp servers **here**.

Command

Download latest NCBI-BLAST+ tools (Linux)

```
wget
https://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/ncbi-
blast-2.16.0+-x64-linux.tar.gz
```

Note

ncbi-blast-2.16.0+ was the latest version as of writing. In the future the URL in the above command will change. You should check **here** for the latest version and adjust the URL accordingly.

Note

If you decide to use the NCBI download to install BLAST+, then you will need to untar the download into a location on your filesystem. The tools will be under a directory named bin. Either you will need to add this directory to your PATH, so that Linux can find the BLAST commands, or you will need to specify the path to your blastn executable using the -blastn option when running the analysis tool in future steps.

7.1 **Check BLAST+ installation**

You can check that the installation of the BLAST+ tools went well by running:

Command

Check blastn version

```
blastn --version
```

Which should report the version of BLAST+ you installed (at least 2.12.0+)

8 **Install FastQC**

FastQC is a tool used for quality control of high-throughput sequence data. It provides a way to assess the quality of raw sequencing data

If you will not follow the quality assessment steps of the protocol, there is no need to install this software.

Most package repositories provide fastqc.

Command

Install fastqc using apt package manager

```
sudo apt-get install fastqc
```

9 **Install Sickle**

Sickle is a software tool designed for quality control of high-throughput sequence data, especially for data generated by Next-Generation Sequencing (NGS) platforms. Its primary use case is trimming low-quality bases and adapter sequences from the ends of sequencing reads

If you will not run the quality assessment steps of the protocol, there is no need to install this software.

Most package repositories provide sickle.

Command

Install sickle using apt package manager

```
sudo apt-get install sickle
```

10 **The TnAtlas tools**

The TnAtlas Python package contains the software tools used in this protocol. TnAtlas can be installed using the Python package installer PIP as follows:

Command

Install TnAtlas using PIP

```
python3 -m pip install tnatlas
```

If the installation is successful then the **tnfind** and **tnmeta** commands should be available:

Command

Check tnfind is installed

```
tnfind
```

Expected result

```
usage: tnfind [-h] [-v] [--sequencing-type {sanger,solexa,illumina}]
              [--input-type {fasta,fastq}] [--input-ext INPUT_EXT] [-o OUTPUT_FILE]
              [-qc [PATH_TO_FASTQC]] [-trim [PATH_TO_SICKLE]] [--trim-quality TRIM_QUALITY]
              [--trim-length TRIM_LENGTH] [--trim-save] [-blastn PATH_TO_BLASTN] [-sam]
              -transposon TRANSPOSON [TRANSPOSON ...] [--transposon-type
TRANSPOSON_TYPE]
              [--transposon-save] [--transposon-word-size TRANSPOSON_WORD_SIZE]
              [--transposon-evalue TRANSPOSON_EVALUE] -genome GENOME [GENOME ...]
              [--genome-type GENOME_TYPE] [--genome-save]
              [--genome-word-size GENOME_WORD_SIZE] [--genome-evalue GENOME_EVALUE]
              [--genome-prefix GENOME_PREFIX] [-config CONFIG]
              input_dir output_dir
tnfind: error: the following arguments are required: input_dir, output_dir, -transposon, -
genome
```

Command

Check tnmeta is installed

```
tnmeta
```

Expected result

usage: tnmeta [-h] [-o OUTPUT_FILE] well_plate_regex result_file metadata [metadata ...]
tnmeta: error: the following arguments are required: well_plate_regex, result_file, metadata

Running the tnfind tool

11 Create a new directory for the results of **tnfind**

Command

new command name

```
mkdir tnfind_results
```

12 Run the **tnfind** command as follows:

Command

new command name

```
tnfind /path/to/reads/directory ./tnfind_results -transposon
/paths/to/transposons.gb -genome /paths/to/reference/genomes.gb
```

where:

- **/path/to/reads/directory** is the path to the directory containing the sequencing reads (prepared in go to step #2)
- **/paths/to/transposons.gb** is the path, or space separated list of paths, to the transposon sequences (prepared in go to step #3)
- **/paths/to/reference/genomes.gb** is the path, or space separated list of paths, to the reference genome sequences (prepared in go to step #1)

tnfind example

13 Create a new directory structure to work in and cd into it:

Command

new command name

```
mkdir tnfind-example && cd tnfind-example
```

14 Download the example sequencing read files, transposon sequences file and reference genome.

Dataset

tnfind example data

NAME

https://saco.csic.es/s/jtwG6EGPCjyJcRk

LINK

You can download the example data from https://saco.csic.es/s/jtwG6EGPCjyJcRk and unzip into a directory called data, or use the command:

Command

new command name

```
wget https://saco.csic.es/s/jtwG6EGPCjyJcRk/download -O data.zip &&
unzip data.zip && rm data.zip
```

15 Make a directory to contain the results

Command

new command name

```
mkdir data/results
```

16 Run tnfind with trimming

Command

new command name

```
tnfind data data/results -transposon data/transposon.gb -genome
data/pputidakt2440.gb -trim
```

Expected result

96 reads from /home/lewis/tnfind-example/data loaded

SE input file: /tmp/tmpy6xbbxx4/records.fastq

Total FastQ records: 96
FastQ records kept: 94
FastQ records discarded: 2

Loading transposons from /home/lewis/tnfind-example/data/transposons.gb
2 transposons loaded
blastn -query /tmp/tmpqoi9a0mu/query.fasta -subject /tmp/tmpqoi9a0mu/target.fasta -parse_deflines -outfmt 5 -evalue 0.01 -word_size 10
Loading genomes from /home/lewis/tnfind-example/data/pputidakt2440.gb
1 genome loaded
blastn -query /tmp/tmpva5irmgr/query.fasta -subject /tmp/tmpva5irmgr/target.fasta -parse_deflines -outfmt 5 -evalue 0.01 -word_size 10
/home/lewis/.local/lib/python3.13/site-packages/tnatlas/cli.py:94: FutureWarning: The behavior of DataFrame concatenation with empty or all-NA entries is deprecated. In a future version, this will no longer exclude empty or all-NA columns when determining the result dtypes. To retain the old behavior, exclude the relevant entries before the concat operation.
  data = pandas.concat([read.dataframe for read in reads.values()])
Saving summary table to /home/lewis/tnfind-example/data/results/results.xlsx

Open the spreadsheet at tnfind-example/data/results/results.xlsx. It should be somewhat similar to:

results.xlsx 14.4KB

Running the tnmeta tool

17 Run the tnmeta command as follows:

Command

new command name

```
tnmeta 'PLATE_WELL_' /path/to/results.xlsx
/path/to/metadata/plate_column.xlsx -o /path/to/output.xlsx
```

where:

- **/path/to/results.xlsx** is the path to an excel file produced by tnfind (for example in go to step #11)
- **/path/to/metadata/plate_column.xlsx** is the path to a metadata file as prepared in go to step #4
- **/path/to/output.xlsx** is the path where you want to create the output of tnmeta

tnmeta example

18 In this example we use tnmeta to annotate the output of the tnfind example. If you have not already, complete the steps for this example at go to step #12

We will annotate using the file:

22CCRAA000_identity_plate.xlsx 18.7KB

Which will look for plates identified as "22CCRAA000", and will create a new column in the results table called "identity". The file is included in the example data, and if you have followed the tnfind example, should already be in the tnfind-example/data directory.

In the tnfind-example directory, run the command:

Command

new command name

```
tnmeta 'PLATE_WELL_' data/results/results.xlsx
data/22CCRAA000_identity_plate.xlsx -o
data/results/metadata_results.xlsx
```

19 You should see a new file has been created at **tnfind-example/data/results/metadata_results.xlsx**, that is similar to:

metaresults.xlsx 42.7KB

Additional Information on tnfind

20 **Basic usage**

tnfind finds possible transposon integration events in a set of sequencing reads and positions them in reference genomes.

Note

tnfind is a command line tool that should be run through a terminal or terminal emulator such as xterm, gnome terminal, Windows powershell or iterm.

At minimum, tnfind requires four arguments:

- **input_dir** is the directory where the sequencing reads in FASTA or FASTQ can be found.
- **output_dir** is the directory into which tnfind will put its results.
- **-transposon** is the GENBANK, FASTA or FASTQ file or files, containing the transposon DNA sequences.
- **-genome** is the GENBANK, FASTA or FASTQ file or files, containing the genomic DNA sequences.

Note

output_dir must already exist for tnfind to work.

Note

Existing files in output_dir are at risk of being overwritten without warning. Be careful when running tnfind over multiple separate datasets using the same output_dir.

A simple example execution of tnfind would look like:

Command

new command name

```
tnfind ./data ./data/results -transposon ./data/transposons.gb -
genome ./data/genomes.gb
```

where in this case input_dir is ./data and contains FASTQ files with extension ab1, output_dir is ./data/results, the file ./data/transposons.gb contains transposon sequences in GENBANK format, and ./data/genomes.gb contains reference genomes in GENBANK format.

21 **Other input formats, types and file extensions**

The following options in combination in order to process input files with different formats, types and file extensions:

- **--input-type**: is the format of the read files and should be either fasta or fastq. The default is fastq.
- **--input-ext**: is the file extension of the read files. This can be anything. The default is ab1 if the --input-type is fastq or fasta if the --input-type is fasta.
- **--sequencing-type**: the type of sequencing used to obtain the reads. This can be either sanger, solexa or illumina. The default is sanger.
- **--transposon-type**: is the format of the file(s) containing the transposon sequences and can be either genbank, fasta or fastq. The default is genbank.

- **--genome-type**: is the format of the file(s) containing the genome sequences and can be either genbank, fasta or fastq. The default is genbank.

For example, if you have FASTQ format reads that have file extension fatstq instead of ab1, you could run:

Command

new command name

```
tnfind ./data ./data/results -transposon ./data/transposons.gb -
genome ./data/genomes.gb --input-ext fastq
```

Or if you do not have an annotated reference genome, and want to instead use a sequence-only FASTA file:

Command

new command name

```
tnfind ./data ./data/results -transposon ./data/transposons.gb -
genome ./data/genomes.fasta --genome-type fasta
```

22 **Quality control report**

Optionally, tnfind can generate a fastqc quality control report for your sequencing reads, that contains information that will be useful if you intend to later trim your reads based on quality.

Passing the option -qc when running tnfind will generate the quality control report. The report will be saved in the output_dir in the file records_fastqc.html, you can open this file with your favourite web browser.

Command

new command name

```
tnfind ./data ./data/results -transposon ./data/transposons.gb -
genome ./data/genomes.gb -qc
```

Note

By default, tnfind looks for a command fastqc on your computer system. If you have installed fastqc somewhere that is not on your PATH, you may also provide the path/to/fastqc to the -qc argument.

Command

new command name

```
tnfind ./data ./data/results -transposon ./data/transposons.gb -
genome ./data/genomes.gb -qc /path/to/fastqc
```

23 **Quality-based trimming**

To ensure that sequencing quality does not affect your results, you may wish to trim your input sequences where quality is low. tnfind can do this using the sickle tool.

Passing the option -trim when running tnfind will trim your reads before processing.

Command

new command name

```
tnfind ./data ./data/results -transposon ./data/transposons.gb -
genome ./data/genomes.gb -trim
```

Optionally, the --trim-length and --trim-quality options can passed to tnfind to control the severity of the trimming.

Command

new command name

```
tnfind ./data ./data/results -transposon ./data/transposons.gb -
genome ./data/genomes.gb -trim --trim-length 30 --trim-quality 25
```

If --trim-length is not given explicitly, its default value is 20. If --trim-quality is not given explicitly, its default value is also 20.

Note

By default, tnfind looks for a command sickle on your computer system. If you have installed sickle somewhere that is not on your PATH, you may also provide the path/to/sickle to the -trim argument.

Command

new command name

```
tnfind ./data ./data/results -transposon ./data/transposons.gb -
genome ./data/genomes.gb -trim /path/to/your/sickle
```

24 **SAM output**

Part of tnfind's processing is a multiple sequence alignment between your sequencing reads and the reference genome(s). Optionally, this multiple sequence alignment may be saved in SAM format.

Passing the option -sam to tnfind will save a multiple sequence alignment in SAM format in the output_dir in the file samgenomes.sam.

Command

new command name

```
tnfind ./data ./data/results -transposon ./data/transposons.gb -
genome ./data/genomes.gb -sam
```

25 **Obtaining help**

There are many more optional arguments that control the behaviour of tnfind.

They can be listed, alongside a short description of each, by running:

Command

new command name

```
tnfind --help
```

26 **Default outputs**

- results.xlsx is an excel spreadsheet containing the summary of the tnfind results.
- tnfind also produces one GENBANK file per input read, which is annotated with the supposed transposon sequence and genome alignments for the read. The genome alignments are also annotated with corresponding genome features and loci, if those are available in the reference genome files.

results.xlsx

This is an excel spreadsheet which contains at least one row for every read in the input to tnfind.

If a particular read has multiple rows, then it was not possible to uniquely identify a position in the reference genome(s) for the transposon integration event. For example, because the integration has occurred within a genomic region which is repeated across the genome(s) (rRNAs for example, or genes with multiple copies, or genes that are shared among genomes).

The columns of the excel describe various different statistics and information about the alignment. There are several groups.

Information about the sequencing read

- The first column is simply an indexing column and can be ignored.
- The **name** column is the ID of the read obtained from the input file.
- The **read length** column is the length of the read (after trimming, if applicable).

- The **multiple candidates** column is true if the read has multiple rows and false otherwise. It can be used to filter uniquely identifiable integration events.

Information about the transposon sequence found

- The **transposon** column contains the name of the supposed transposon sequence.
- The **transposon evalue** column contains the evalue, reported by blastn, of the alignment of the read to the transposon sequence.
- The **transposon bit score** column contains the bit score, reported by blastn, of the alignment of the read to the transposon sequence.
- The **transposon identity** column contains the identity score, reported by blastn, of the alignment of the read to the transposon sequence. 1 is 100% identity.
- The **transposon length** column indicates the length of the aligned part of the transposon sequence.
- The **transposon start** column indicates the base pair position in the read where the alignment to the transposon sequence begins.
- The **transposon end** column indicates the base pair position in the read where the alignment to the transposon sequence ends.

These columns are empty if no transposon sequence is found in the read.

Information about the position in the genome

- The **genome offset** column indicates the distance in base pairs between the end of the transposon alignment and the beginning of the genome alignment.
- The **gap sequence** is the read sequence (if any), between the end of the transposon alignment and the beginning of the genome alignment.
- The **insertion locus** is the name of the locus (if any) in the genome where the integration event is supposed to have occurred. This name comes from the annotated reference genome.
- The **insertion gene** is the name of the gene into which the transposon sequence was integrated. This name comes from the annotated reference genome. If there is no such annotation this column shows "None"
- The **sequence type** is the annotated type of the locus into which the integration has occurred. Again, this is taken from annotations on the reference genome.
- The **insertion strand** is the strand orientation of the locus into which the integration has occurred.
- The **insertion product** is the product annotation of the locus into which the integration has occurred.

These columns are empty if there is no alignment to the genome, or if genome annotations are missing.

Information about the genome alignment

- The **genome** column contains the ID of the reference genome to which the read has been aligned.
- The **genome alignment evalue** columns contains the evalue, reported by blastn, of the alignment of the read to the genome sequence.
- The **genome alignment bit score** column contains the bit score, reported by blastn, of the alignment of the read to the genome sequence.
- The **genome alignment identity** column contains the identity score, reported by blastn, of the alignment of the read to the genome sequence. 1 is 100% identity.
- The **genome alignment length** column contains the length in base pairs of the part of the read that was aligned to the genome.
- The **genome alignment mismatch** column contains the number of mismatches, as reported by blastn, in the alignment between the read and the genome.
- The **genome alignment gaps** column contains the number of gaps, as reported by blastn, in the alignment between the read and the genome.
- The **genome start** column indicates the base pair position in the reference genome of the start of the alignment between the read and the genome.
- The **genome end** column indicates the base pair position in the reference genome of the start of the alignment between the read and the genome.
- The **genome strand** column indicates the strand orientation on the genome of the alignment between the read and genome.
- The **genome loci** column is a list of all the genomic loci contained within the genome alignment (as opposed to the single locus at the integration position reported in the insertion locus column).
- The **genome prefix** column displays the sequence corresponding to the first 9 base pairs of genomic DNA contained in the read (the start of the read sequence that aligns with the genome). The size of the sequence displayed can be changed with the --genome-prefix option.
- The **genome alignment** column shows the quality of the alignment of the genome prefix sequence with the genome. Exact matches are indicated with "|". Mismatches are indicated with ".". Gaps are show as "-".

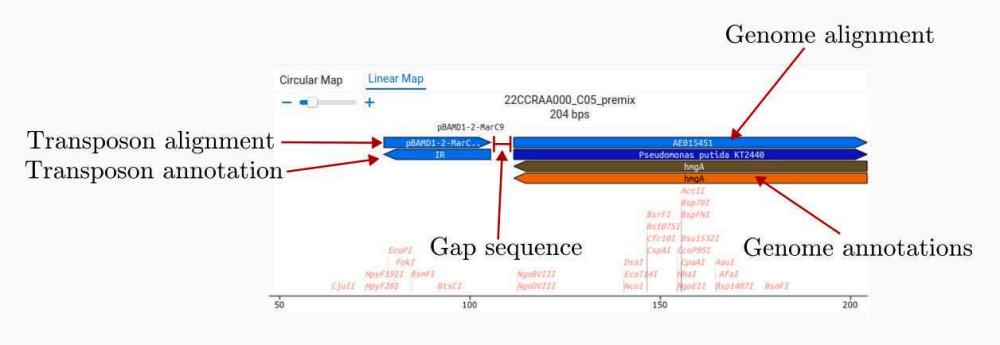
These columns are empty if there is no alignment to the genome.

Annotated reads

tnfind produces an annotated GENBANK file (file names ending in .processed.gb) for each input read that can be used to visually inspect the positions and alignments found to the both the transposon sequences and the genomes.

These files can be opened using your favourite sequence viewer to see the arrangement of transposon and genomic sequences that tnfind thinks is most probable for each read. Genomic sequences are also annotated with features from their respective reference genomes.

An example:



An example of the GENBANK output from tnfind. In this case, the read 22CCRRAA000_C05 is annotated with a transposon sequence alignment, plus the IR (Inverted repeat) feature. There is a small gap between the end of the transposon sequence and the genomic sequence, which comes from the genome AE015451.2 and contains the gene hmgA. In this case, the read aligned with the plus strand of AE015451.2, but the hmgA gene is coded for on the minus strand.

Additional Information on tnmeta

27 It is sometimes useful to add additional data to the results.xlsx spreadsheet. The tnmeta tool can be used to attach metadata to each read, based on the plate and well from which the read was sampled.

> Note
>
> tnmeta is a command line tool that should be run through a terminal or terminal emulator such as xterm, gnome terminal, Windows powershell or iterm.

The tnmeta tool requires 3 arguments.

- **well_plate_regex** is a regular expression that will be used to extract the plate and well number from the read IDs.
- **result_file** is the file which is to be annotated with the metadata.
- **metadata** are the files which contain the maps of metadata.

By default, **result_file** is overwritten by tnmeta. If you don't want to overwrite **result_file** you can pass the option **-o <name/of/new/file>** to save the annotated results to a separate file.

Note

If you already have **result_file** open in your spreadsheet software, you may encounter a permission denied error when trying to run tnmeta. Close the **result_file** spreadsheet first and try again.

28 **Naming the metadata file**

The name of the metadata file will be used to determine to which plate the metadata corresponds, and the name of the new column that should contain the metadata in the results table.

The name of the file must begin with an alphanumeric plate identifier, followed by an underscore, followed by an alphanumeric column name.

A file "PLATE_COLUMN.xlsx" will look for reads from the plate called PLATE, and add metadata to reads in a new column called COLUMN.

You should rename your metadata files accordingly.

29 **Well and plate regular expression**

The **well_plate_regex** argument is used to match read IDs to a specific plate, and to extract the well position from the read IDs. The regular expression should contain the keywords **PLATE** and **WELL**, which indicate the positions of the plate identifier and well position in the read ID. All other parts of the regular expression are interpreted using the Python **re regular expression syntax.**

For example, if we have read IDs of the form "22CCRAA000_A01_premix", where "22CCRAA000" is the plate identifier and "A01" is the well position, we could use the regular expression:

```
PLATE_WELL_
```

Which means that tnmeta should expect the plate identifier to come first, then an underscore, then the well position, followed by another underscore.

Note

To avoid problems due to shell expansion of regular expression wildcards, it is a good idea to pass the **well_plate_regex** argument inside quotes, for example, 'PLATE_WELL_'

Protocol references

https://doi.org/10.1093/synbio/ysad012
https://doi.org/10.1021/acssynbio.3c00397