TEtrimmer Manual

Last updated: July 1, 2024
Author: Hang Xue; Jiangzhao Qian

TEtrimmer Github repository

1. Getting started

1.1 Installation

TEtrimmer can be installed by 1. Conda, 2. Singularity, or 3. Docker.

1. **Conda**

   We highly recommend installation with `mamba`, as it is much faster. On some hosts, Python 3.10 installation might be required as well.

   ```
   conda create --name TEtrimmer
   conda install -c conda-forge mamba
   conda activate TEtrimmer
   mamba install bioconda::tetrimmer
   TEtrimmer --help

   # If you encounter "ClobberError" or "ClobberWarning", don't
   worry! wait until it is finished!
   # The Error or Warning could be like this:
   ClobberError: This transaction has incompatible packages due to a
   shared path.
     packages: bioconda/osx-64::blast-2.5.0-boost1.64_2,
   bioconda/osx-64::rmblast-2.14.1-hd94f91d_0
     path: 'bin/blastx'
   ```

   or you can install by the provided .yml file (Only for Linux).

   ```
   conda install -c conda-forge mamba
   git clone https://github.com/qjiangzhao/TEtrimmer.git
   mamba env create -f [path to/TEtrimmer_env_for_linux.yml]
   python <path to TEtrimmer>/TEtrimmer.py --help
   ```

   or See required dependencies TEtrimmer_dependencies

2. **Singularity**

   ```
   singularity pull \
   docker://quay.io/biocontainers/tetrimmer:1.4.0--hdfd78af_0

   # Specify path to a PFAM database using --pfam_dir. If
   <your_path_to_store_PFAM_database> doesn't contain PFAM database,
   TEtrimmer can automatically download PFAM to
   <your_path_to_store_PFAM_database>

   singularity exec --writable-tmpfs \
   ```

```
--bind <your_path_contain_genome_file>:/genome \
--bind <your_path_contain_input_TE_library_file>:/input \
--bind <your_output_path>:/output \
--bind <your_path_to_store_PFAM_database>:/pfam \
<your_path_to_sif_file>/tetrimmer_1.4.0--hdfd78af_0.sif \
TEtrimmer \
-i /input/<TE_library_name.fasta> \
-g /genome/<genome_file_name.fasta> \
-o /output \
--pfam_dir /pfam \
-t 20 --classify_all
```

3. **Docker**

```
docker pull quay.io/biocontainers/tetrimmer:1.4.0--hdfd78af_0
docker run -it --name TEtrimmer -v <bind_your_path>:/data
quay.io/biocontainers/tetrimmer:1.4.0--hdfd78af_0
```

1.2 Test run

For users running TEtrimmer for the first time, we recommend testing the software with example test files. This practice ensures TEtrimmer, along with its associated databases, is correctly installed and capable of executing without errors. Download example test files from Github: test_input.fa and test_genome.fasta.

During the first run, a Pfam database (around 1.6 Gb) will be automatically downloaded to the default path (unless the user specifies a custom database or a different download path by --pfam_dir). This step requires a network connection and might take around 10 mins. The test run typically takes 5 to 10 minutes, depending on your computing setup.

Test run command:

```
TEtrimmer -i <path to test_input.fa> \
          -g <path to test_genome.fasta> \
          -o <output_directory> \
          -t 10 --classify_all
```

If the above command is run successfully, your terminal will look like below:

Firstly, the Pfam database will be downloaded.

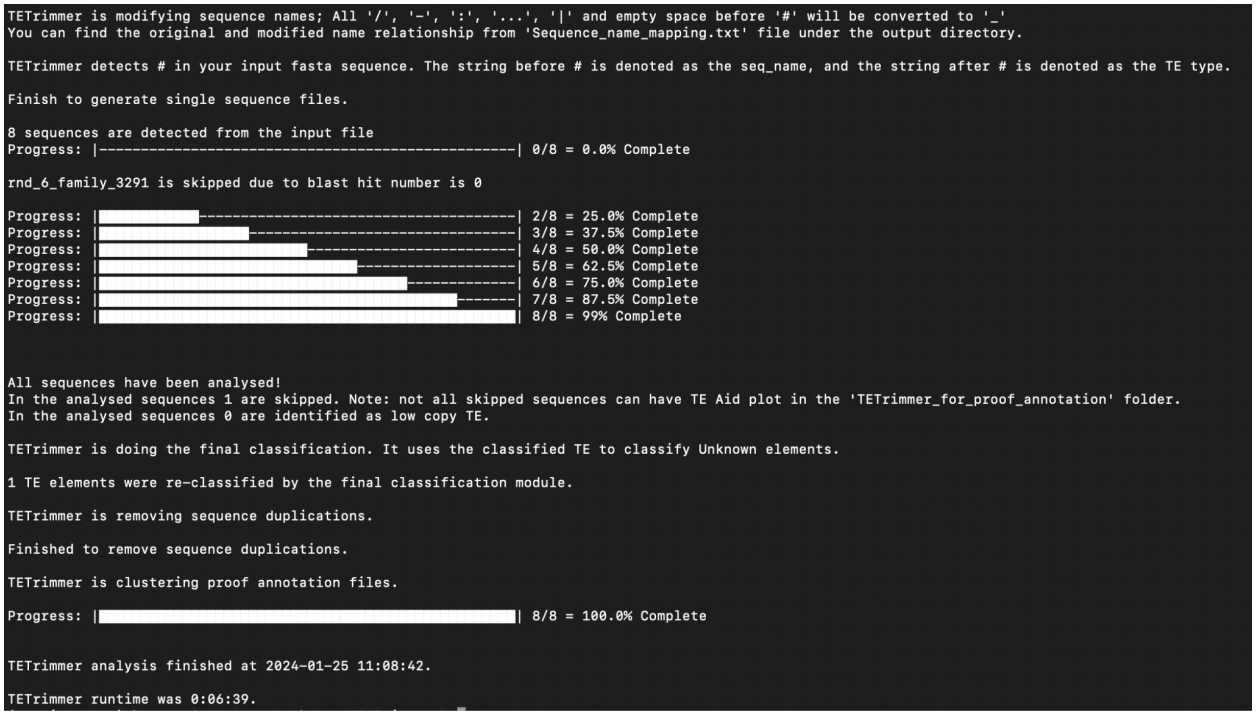
```
Pfam-A.hmm not found. Downloading... This might take some time. Please be patient


Pfam-A.hmm is downloaded and unzipped. Pfam database was stored in
/Users/hangxue/Documents/Wildermuth_Lab/TE_Trimmer/pfam_database


Pfam-A.hmm.dat not found. Downloading... This might take some time. Please be patient


Pfam-A.hmm.dat is downloaded and unzipped. Pfam database was stored in
/Users/hangxue/Documents/Wildermuth_Lab/TE_Trimmer/pfam_database
```

Finally, the progress bar should show 100% completeness.



Note: It might take a long time from 99% to 100% for your genome. TEtrimmer performs de-duplication and clustering via CD-EST-HIT which can be computationally intensive when dealing with many sequences.

1.3 Understand TEtrimmer output files

Your output directories should look the same as the following:

- 📁Classification_and_deduplication
- 📁Multiple_sequence_alignment
- 📁Single_fasta_files
- 📁TE_Trimmer_for_proof_annotation
  - 📁Annotation_perfect
  - 📁Annotation_good
  - 📁Annotation_recommend_check
  - 📁Anotattion_need_check
  - 📁Clustered_proof_annotation
    - 📁Cluster1
    - 📁Cluster2
    - 📁Cluster3
    - ……
  - 📁TE_low_copy
  - 📁TE_skipped
  - 📁TEtrimmer_proof_anno_GUI
    - 📄annoGUI.py
    - ……
- 📄Sequence_name_mapping.txt
- 📄summary.txt
- 📄TE_Trimmer_consensus.fasta
- 📄TE_Trimmer_consensus_merged.fasta

You can compare your test results with the example test_output on GitHub to make sure TEtrimmer is running successfully on your setup. Note: the results of each run will be slightly different due to the randomness in alignment and clustering steps. Explanations for each file are as below.

📄**summary.txt** - summary file. This file contains a summary of each processed sequence.

summary

input_name	consensus_name	blast_hit_n	cons_MSA_seq_n	cons_full_blast_n	input_length	cons_length	input_TE_type	reclassified_type	terminal_repeat	low_copy	evaluation	status
rnd_6_family_3291	rnd_6_family_3291	0	NaN	NaN	447	447	Unknown	Unknown	FALSE	FALSE	NaN	skipped
rnd_1_family_667	rnd_1_family_667	27	12	9	1382	5363	Unknown	LTR/Copia	FALSE	FALSE	Need_check	processed
rnd_6_family_2561	rnd_6_family_2561	16	16	7	2508	2209	Unknown	DNA/TcMar-Fot1	FALSE	FALSE	Need_check	processed
rnd_1_family_587	rnd_1_family_587	41	12	9	771	2856	LTR/Copia	LTR/Copia	FALSE	FALSE	Need_check	processed
rnd_1_family_587	rnd_1_family_587_01	41	12	6	771	2826	LTR/Copia	LTR/Copia	FALSE	FALSE	Need_check	processed
rnd_5_family_182	rnd_5_family_182	57	54	31	433	395	Unknown	Unknown	FALSE	FALSE	Reco_check	processed
rnd_1_family_188	rnd_1_family_188	31	26	21	6453	6467	LTR/Gypsy	LTR/Gypsy	LTR	FALSE	Good	processed
rnd_1_family_588	rnd_1_family_588	534	84	416	306	3214	DNA/MULE-MuDR	DNA/MULE-MuDR	FALSE	FALSE	Reco_check	processed
rnd_1_family_470	rnd_1_family_470	157	50	42	5760	5847	Unknown	LTR/Gypsy	LTR	FALSE	Perfect	processed
rnd_1_family_470	rnd_1_family_470_01	157	17	26	5760	5051	Unknown	LTR/Gypsy	FALSE	FALSE	Need_check	processed

input_name: the sequence header of the original input sequence with converted special characters. If # is present in the original header, the string after # is used for the TE classification, and the string before # is kept for naming the sequence.

consensus_name: the sequence header of the processed consensus sequence. In the case where an input sequence can yield multiple processed consensus sequences, the subsequent "consensus_name" will be appended by a numerical identifier at the end. (eg. "rnd_1_family_470" and "rnd_1_family_470_01" are two processed consensus sequences derived from the same input sequence "rnd_1_family_470").

blast_hit_n: the blast hit number of the original input sequence against your genome file.

cons_MSA_seq: sequence number of the multiple sequence alignment (MSA) used for generating the TE consensus sequence.

cons_full_blast: the full blast hit number of the processed consensus sequence. Full blast means the blast hits must cover more than 90% of the query sequence.

input_length: the length of the original input sequence.

cons_length: the length of the TEtrimmer processed consensus sequence.

input_TE_type: the TE class of the original input sequence if known. Otherwise, it will be "Unknown".

reclassified_TE_type: the TE class of the processed consensus sequence either inherited from the original input sequence or reclassified by TEtrimmer. For example, "rnd_6_family_2561" is reclassified as "DNA/TcMar-Fot1" after processing.

terminal_repeat: the presence of either TIR or flanking LTR at both ends of the processed consensus sequence.

low_copy: whether the input sequence is considered as a low copy TE element. TE will be denoted as low copy when its BLASTn hit number is less than 10, but with more than 2 full-length BLASTn hits and terminal repeat (LTR or TIR).

evaluation: the processed consensus sequence is evaluated based on the parameters shown below. The "Perfect" and "Good" ones are high-quality TE consensus sequences that require no or minimal modifications while the "Reco_check" and "Need_check" might need additional manual proof curation by the provided TEtrimmer GUI.

Criteria	Terminal_repeat	Classified	cons_MSA_seq	cons_full_blast	if_PFAM
Perfect	TRUE	TRUE	>=30	>=5	TRUE
Good	TRUE	Not_required	>=10	>=2	Not_required
Reco_check	Not_required	Not_required	>=20	>=2	Not_required
Need_check	Not_required	Not_required	Not_required	Not_required	Not_required

Status: skipped or processed. If a sequence is skipped, it is not included in the "TE_Trimmer_consensus.fasta" file. The input TE sequence will be skipped when the copy number is less than 10 and does not belong to "low copy" TEs.

📄**TE_Trimmer_consensus.fasta**: fasta file of TEtrimmer processed TE consensus sequences before de-duplication.

📄**TE_Trimmer_consensus_merged.fasta**: fasta file of processed TE consensus sequences file after de-duplication.

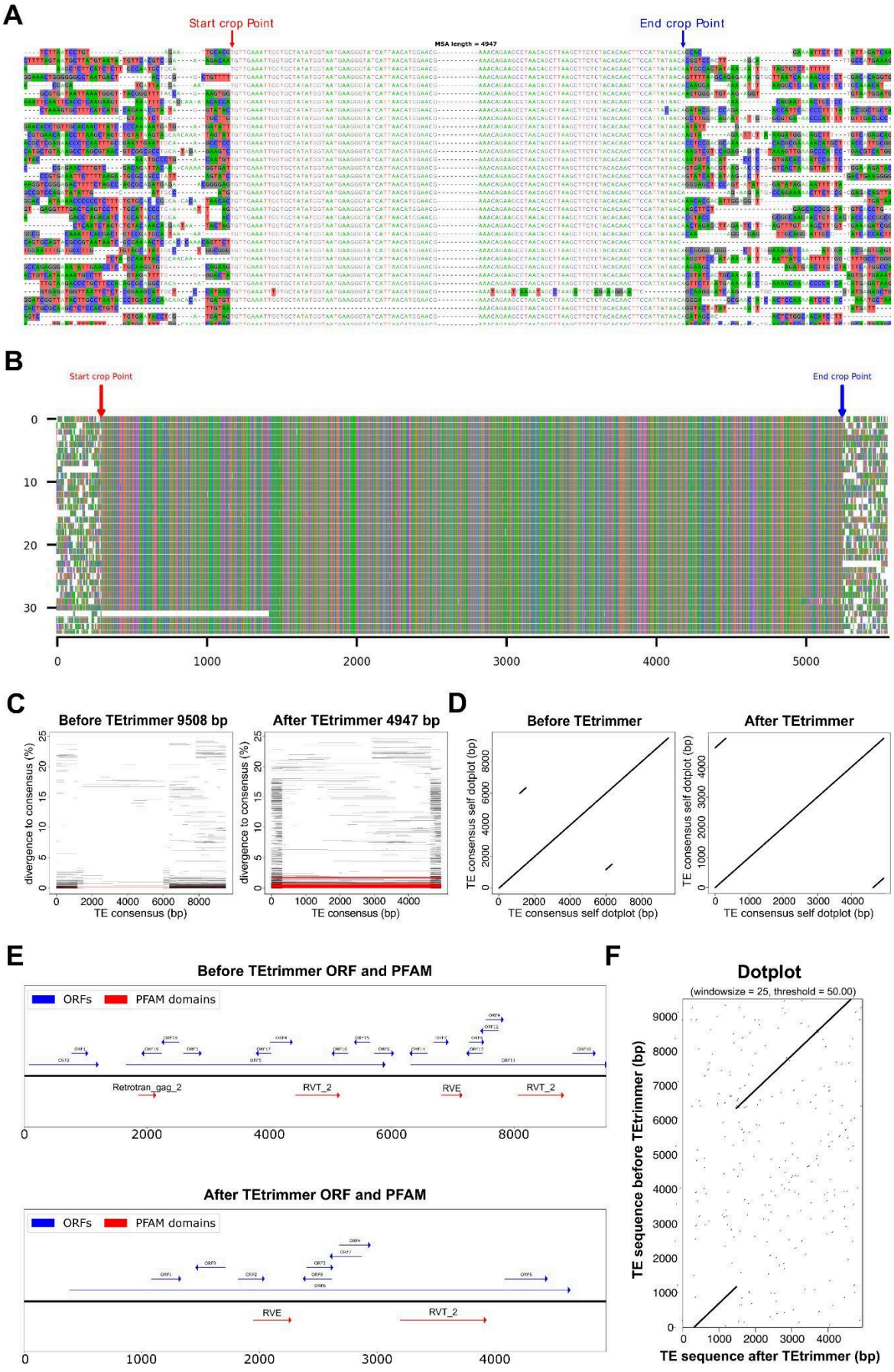
📄**Sequence_name_mapping.txt**: text file maps TEtrimmer modified input fasta sequence names with original names. Any occurrence of '/', '-', ':', '...', '|' and empty spaces before '#' will be converted to '_'

📁**Multiple_sequence_alignment**:
📄**error_file.txt** - stor all error messages, only present when errors were found.
All intermediated analysis files will be kept when the "--debug" option is enabled.

📁**Single_fasta_files**: All sequences in the input file will be separated into single fasta files and be stored here to allow multithreads analysis.

📁**Classification_and_deduplication**:
📄_round*.fasta_clstr: merged sequences after round* cd-hit-est
📄 temp_TE_Trimmer_classified_consensus.fasta: processed consensus sequences that have known TE classification
📄 temp_TE_Trimmer_unknown_consensus.fasta: processed consensus sequences that have "Unknown" TE classification
📁 temp_repeatmasker_classification: result files after running RepeatMasker to classify unknown sequences (temp_TE_Trimmer_unknown_consensus.fasta) by classified sequences (temp_TE_Trimmer_classified_consensus.fasta)
📄 TEtrimmer_consensus_no_low_copy.fasta: TE consensus library before deduplication but don't include low copy element. Used to cluster manual proof annotation files
📄 TEtrimmer_consensus_no_low_copy.fasta_cd.fa: cd-hit-est fasta output
📄 TEtrimmer_consensus_no_low_copy.fasta_cd.fa.clstr: cd-hit-est output containing clustering information.

📁**TE_Trimmer_for_proof_annotation**: This folder contains files that can be used for additional manual proof annotation.
📁**Annotation_perfect**: Four files are associated with each output TE sequence.
📄**TE_name.cluster.fa**: MSA file before clustering
📄**TE_name.raw.fa:** MSA file after clustering but before cleaning
📄**TE_name.fa**: MSA file used for consensus sequence generation
📄**TE_name.pdf**: Plot file used to evaluate output. See the explanation below**.**


A
Start crop Point
End crop Point
MSA length = 4947
B
Start crop Point
End crop Point
0
10
20
30
0
1000
2000
3000
4000
5000
C
Before TEtrimmer 9508 bp
After TEtrimmer 4947 bp
divergence to consensus (%)
TE consensus (bp)
D
Before TEtrimmer
After TEtrimmer
TE consensus self dotplot (bp)
E
Before TEtrimmer ORF and PFAM
ORFs
PFAM domains
Retrotran_gag_2
RVT_2
RVE
After TEtrimmer ORF and PFAM
F
Dotplot
(windowsize = 25, threshold = 50.00)
TE sequence before TEtrimmer (bp)
TE sequence after TEtrimmer (bp)

Figures A, B: these two figures give you a visual representation of the multiple sequence alignment (MSA) of the processed consensus sequences. Ideally, the MSA should have clear boundaries at both ends.
Figure C, D: this part gives you the comparison between the processed consensus sequence and its original input sequences using TE-Aid plots. Left panel: the genomic hits with divergence to consensus. The red color indicated high-quality hits. Right panel: a self dot-plot. (https://github.com/clemgoub/TE-Aid)
Figure E: this part gives you the comparison of the predicted open reading frame (ORF) and PFAM domains between consensus sequences before and after TEtrimmer analysis.
Figure F: This figure is the dot plot between the input sequence and processed consensus sequence. The x-axis represents the TEtrimmer processed TE consensus sequence and the y-axis is the original input sequence.

📁**Annotation_good**: same as the previous folder

📁**Annotations_check_recommended**: same as the previous folder

📁**Annotations_check_required**: same as the previous folder

📁**Clustered_proof_annotation**: The folder copies all files from “Annotation_perfect”, “Annotation_good”, “Annotation_check_recommended”, and “Annotation_check_required" and groups them into different clusters based on TE consensus sequence similarity. You can do your manual proof curation based on this folder. Theoretically, you only need to choose one set of consensus files from each cluster folder.

📁**TE_low_copy**
- 📄**TE_name.fa**: Input sequence that is classified as a low-copy element
- 📄**TE_name.pdf**: Plots file helps to assess if it is a true low-copy TE

📁**TE_skipped**
- 📄**TE_name.pdf**: Plots file helps to assess if it is a true skipped TE

📁**TEtrimmer_proof_anno_GUI**
- 📄annoGUI.py: Graphical user interface tool to help manual proof curation. Use “python ./annoGUI.py -g [genome.fa]” to start it.

2. Obtain input files

2.1 Genome file

Select a genome database or repository that hosts the genetic information for your chosen organism. Some commonly used databases include: National Center for Biotechnology Information (NCBI), Ensembl, UCSC Genome Browser, and European Bioinformatics Institute (EBI). Downloading your genome as a FASTA file (.fa or .fasta).

2.2 TE consensus library

TEtrimmer uses the TE consensus library from de novo TE annotation tools, like RepeatModeler or EDTA, as input. For this reason, you have to run RepeatModeler or other TE annotation software first. You can find details about how to use RepeatModeler at https://www.repeatmasker.org/RepeatModeler/ (Flynn et al. 2020) or EDTA (Su et al. 2021). TEtrimmer package already includes RepeatModeler. Below is an example command of running RepeatModeler. (-LTRStruct option is highly recommended to use. This increases the running time of RepeatModeler but you will get more intact TEs at the end).

```
BuildDatabase -name <genome> <genome.fa>

RepeatModeler -database <genome> -threads 20 -LTRStruct >& run.out &
```

The RepeatModeler output file <genome_families.fa> can be used as the input TE consensus library for TEtrimmer.

3. Run TEtrimmer

3.1 Hardware requirement

System: Linux, macOS

RAM:

- For HPC Linux users, enough RAM needs to be assigned. We highly recommend running TEtrimmer on HPC with at least 40 threads and assigning at least 5 GB RAM to each thread.

Threads	RAM
40	200 GB
100	600 GB

- Windows WSL and macOS users can use Virtual Memory. Simply assign 20 threads to push the CPU to its limits. We did tests on a Macbook Pro (2020 M1 chip, 16 GB RAM) and compared with HPC, you can find the running time here:

Query sequence number	Platform	Threads	RAM	Running time
1700	Macbook Pro M1	20	16 GB + Virtual Memory	50 hours
1700	HPC	40	200 GB	5 hours

3.2 Basic run examples

Note: if your <output_directory> is not empty, TEtrimmer will create a new folder under it named using the following format “TEtrimmer_output_yyyymmdd_hhmmss” (eg. TEtrimmer_output_20240108_125418)

```
TEtrimmer -i <TE_consensus_library.fa> \
          -g <genome.fa> \
          -o <output_directory> \
          -t 20  \
          --classify_all
```

If you want to continue the analysis based on previous unfinished results in the same directory:

```
TEtrimmer -i <TE_consensus_library.fa> \
          -g <genome.fa> \
          -o <directory_to_unfinished_results> \
          -t 20  \
          --classify_all \
          --continue_analysis
```

If you are combining files from different sources for the input file, we recommend removing duplicate sequences during processing. This step can potentially save overall run time.

```
TEtrimmer -i <TE_consensus_library.fa> \
          -g <genome.fa> \
          -o <output_directory> \
          -t 20  \
          --classify_all \
          --dedup
```

3.3 Basic options

The basic options to run TEtrimmer are as follows:

--input_file, -i
: Required
string: path to TE consensus library (FASTA format). Use the output from RepeatModeler or EDTA et al.

--genome_file, -g
: Required
string: Path to genome file (FASTA format).

--output_dir, -o
: Default: current working directory
string: Path to the output directory

--preset, -s
: Default: conserved
string: choose one preset config from below (see more info in section 3.5):
conserved
divergent

--num_threads, -t
: Default: 10
int: the thread numbers used for TEtrimmer.

--classify_unknown
Default: False
boolean: Use RepeatClassifier to classify the processed consensus sequence if the input sequence is not classified or is unknown.

--classify_all
Default: False
boolean: Use RepeatClassifier to classify every processed consensus sequence. WARNING: it will take a longer time.

--continue_analysis, -ca
Default: False
boolean: continue from the previous unfinished TEtrimmer run and use the same output directory.

--dedup
Default: False
boolean: remove duplicate sequences in the input file. Recommend using this option if the user is combining results from output files from different de novo TE annotatils tools. This will help reduce total running time without losing much info. (95% length and 95% similarity)

--genome_anno, -ga
Default: False
boolean: perform genome TE annotation using RepeatMasker with the TEtrimmer curated TE libraries.

--help
Default: False
boolean: show help message and exit.

3.4 Advanced options

--debug
Default: False
boolean: debug mode. This will keep all raw files. WARNING: a lot of files will be kept, especially in the Multiple_sequence_alignment folder.

--pfam_dir, -pd
Default: None
string: Pfam database directory. TEtrimmer will download the database automatically. Only turn on this option if you want to either specify a different default download path, or use a local PFAM database, or the automatic download fails.

--cons_thr
Default: 0.8
float: The minimum level of agreement required at a given position in the alignment for a consensus character to be called. Positions with agreement less than the threshold will be represented by the ambiguous character "N" in the consensus sequence.

--mini_orf

Default: 200

int: the minimum ORF length that will be predicted by TEtrimmer.

--max_msa_lines

Default: 100

int: the maximum number of sequences to be included in a multiple sequence alignment. We do not recommend increasing this number.

--top_mas_lines

Default: 100

int: if the sequence number of multiple sequence alignment (MSA) is greater than <max_msa_lines>, TEtrimmer will first sort sequences by length and choose <top_msa_lines> number of sequences. Then, TEtrimmer will randomly select sequences from all remaining BLAST hits until <max_msa_lines> sequences are found for the multiple sequence alignment.

--min_seq_num

Default: 10

int: the minimum blast hit number required for the input sequence. If the blast hit number is below <min_seq_num>, this sequence will be skipped. Additionally, it serves as the minimum requirement for the number of sequences needed to form a valid cluster during multiple sequence alignment. If the blast hit or number of sequences in each cluster is below <min_seq_num>, this sequence will also be skipped. A higher value of this number might lead to more low copy TE skipped but more accurate results. We do not recommend decreasing this number.

--min_blast_len

Default: 150

int: The minimum sequence length for blast hits to be included for further analysis.

--max_cluster_num

Default: 2

int: The maximum number of clusters assigned in each multiple sequence alignment. Each multiple sequence alignment can be grouped into different clusters based on alignment patterns. TEtrimmer will sort clusters by size and choose the top <max_cluster_num> of clusters for further analysis. WARNING: using a larger number will potentially result in more accurate consensus results but will significantly increase the running time. We do not recommend increasing this value to over 5.

--ext_thr

Default: 0.7

Float: the threshold to call “N” at a position. For example, if the most conserved nucleotide in a MSA column has proportion smaller than <ext_thr>, a “N” will be called at this position. Used with <ext_check_win>. The lower the value of <ext_thr>, the more likely to get longer the extensions on both ends. You can try reducing <ext_thr> if TEtrimmer fails to find full-length TEs.

--ext_check_win

Default: 150

int: the check windows size during defining start and end of the consensus sequence based on the multiple sequence alignment. Used with <ext_thr>. If <ext_check_win> bp at the end of multiple sequence alignment has “N” present (ie. positions have similarity proportion

smaller than <ext_thr>), the extension will stop, which defines the edge of the consensus sequence.

--ext_step
Default: 1000
int: the number of nucleotides to be added to the left and right ends of the multiple sequence alignment in each extension step. TE_Trimmer will iteratively add <ext_step> nucleotides until finding the TE boundary or reaching <max_ext>. For organisms with high divergence such as in rice, we recommend increasing this number (eg. 1500).

--max_ext
Default: 7000
int: the maximum extension in nucleotides at both ends of the multiple sequence alignment. For organisms with high divergence such as in rice, we recommend increasing this number (eg, 9000).

--gap_thr
Default: 0.4
float: If a single column in the multiple sequence alignment has a gap proportion larger than <gap_thr> and the proportion of the most common nucleotide in this column is less than <gap_nul_thr>, this column will be removed from the consensus.

--gap_nul_thr
Default: 0.7
float: the nucleotide proportion threshold for keeping the column of the multiple sequence alignment. Used with the <gap_thr> option. i.e. if this column has <40% gap and the portion of T (or any other) nucleotide is >70% in this particular column, this column will be kept.

--crop_end_div_thr
Default: 0.8
float: the crop end by divergence function will convert each nucleotide in the multiple sequence alignment into a proportion value. This function will iteratively choose a sliding window from each end of each sequence of the MSA and sum up the proportion numbers in this window. The cropping will continue until the sum of proportions is larger than <crop_end_div_thr>. Cropped nucleotides will be converted to -.

--crop_end_div_win
Default: 20
int: the window size used for the end-cropping process. Used with <crop_end_div_thr> option.

--crop_end_gap_thr
Default: 0.1
float: the crop end by gap function will iteratively choose a sliding window from each end of each sequence of the MSA and calculate the gap proportion in this window (i.e. the number of columns that are gaps in the window size of <crop_end_gap_win>). The cropping will continue until the sum of gap proportions is smaller than <crop_end_gap_thr>. Cropped nucleotides will be converted to -.

--crop_end_gap_win
Default: 250
int: the window size used to crop end by gap, used with <crop_end_gap_thr> option.

--start_patterns
Default: TG
string: LTR elements always start with a conserved sequence pattern. TE Trimmer searches the beginning of the consensus sequence for these patterns. If the pattern is not found, it will extend the search of <start_patterns> to up to 15 nucleotides from the beginning of the consensus sequence and redefine the start of the consensus sequence if the pattern is found. Note: The user can provide multiple LTR start patterns in a comma-separated list, like: TG,TA,TC (no spaces; the order of patterns determines the priority for the search).

--end_patterns
Default: CA
string: LTR elements always end with a conserved sequence pattern. TE Trimmer searches the end of the consensus sequence for these patterns. If the pattern is not found, it will extend the search of <end_patterns> to up to 15 nucleotides from the end of the consensus sequence and redefine the end of the consensus sequence if the pattern is found. Note: The user can provide multiple LTR end patterns in a comma-separated list, like: CA,TA,GA (no spaces; the order of patterns determines the priority for the search).

3.5 Default parameters associated with different organisms

The preset parameters below still have room for improvement and there is no guarantee that one of them will be optimal for your organism. The default (conserved) setting is considered better if your organism has relatively more conserved TE elements, i.e. the MSA of blast hits of your input sequence look clean and conserved (eg. the screenshot of TE_name.pdf in Section 1.3). The divergent setting has a longer extension period of TE elements and lower conservation threshold when deciding the boundary of TE elements. The setting for divergent TE elements will take longer to finish the TEtrimmer run than the conserved one.


```
"conserved": {
    "cons_thr": 0.8,
    "max_msa_lines": 100,
    "top_mas_lines": 100,
    "min_seq_num": 10,
    "min_blast_len": 150,
    "max_cluster_num": 2,
    "ext_thr": 0.7,
    "ext_step": 1000,
    "max_ext": 7000,
    "gap_thr": 0.4,
    "gap_nul_thr": 0.7,
    "crop_end_div_thr": 0.8,
    "crop_end_div_win": 40,
    "crop_end_gap_thr": 0.1,
    "crop_end_gap_win": 250,
    "start_patterns": "TGT",
```

```
  "end_patterns": "ACA",
  "mini_orf": 200,
  "ext_check_win": 150
 },
 "divergent": {
  "cons_thr": 0.8,
  "max_msa_lines": 100,
  "top_mas_lines": 100,
  "min_seq_num": 10,
  "min_blast_len": 150,
  "max_cluster_num": 2,
  "ext_thr": 0.6,
  "ext_step": 2000,
  "max_ext": 8000,
  "gap_thr": 0.4,
  "gap_nul_thr": 0.7,
  "crop_end_div_thr": 0.7,
  "crop_end_div_win": 40,
  "crop_end_gap_thr": 0.1,
  "crop_end_gap_win": 250,
  "start_patterns": "TG",
  "end_patterns": "CA",
  "mini_orf": 200,
  "ext_check_win": 150
 },
```

4. Understand additional output files

See common output files in Section 1.3. We provide a brief explanation of additional output files below when the “debug” option is used. For further explanation, feel free to email the authors.

📁**Multiple_sequence_alignment** - All raw files will be kept in this folder when <debug > is enabled. Explanations of selected files are shown below.

For each input sequence, it might have the associated files below

- 📄*.b: blast result file
- 📄*.b.bed: bed file converted from blast result file
- 📄*.bed_u: bed file after removing identical lines
- 📄*.bed_uf.bed: bed file after filter (only choose the top 100 long sequence for multiple sequence alignment)
- 📄*.bed-_*_*_n.*: bed file or alignment file with left and right extension number (eg. bed_1000_2000 means left 1000 extension and right 2000 extension).
- 📄 *_n.fa: fasta file, use the bed file name column as sequence header
- 📄 *_bcln.fa: bed file that has been cleaned. All letters in the fasta sequence not belonging to [AGCTagct] will be deleted.
- 📄 *_aln.fa – aligned fasta file
- 📄 *_aln.fa_gs – aligned fasta file after gap removing
- 📄 *.fa_pat_MSA.fa – pat means pattern. Distinct columns in the MSA are extracted and used for clustering.
- 📄 *.iqtree – iqtree result
- 📄 *._g_*.bed – bed file for cluster *
- 📄*_fm.bed: final multiple sequence alignment

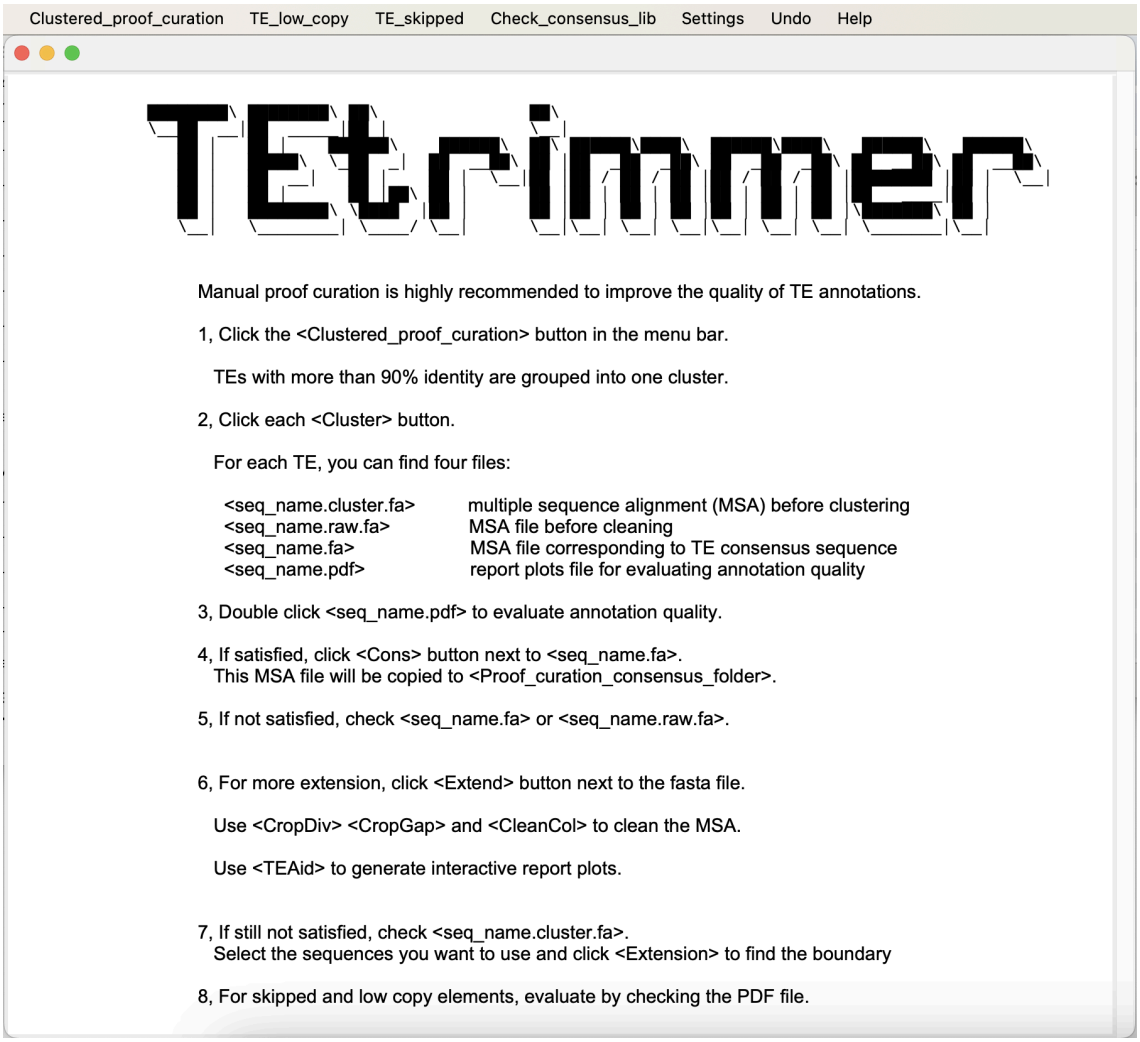
📁HMM - This folder is used to store Hidden Markov Model file. Only created when <-hmm> is enabled.

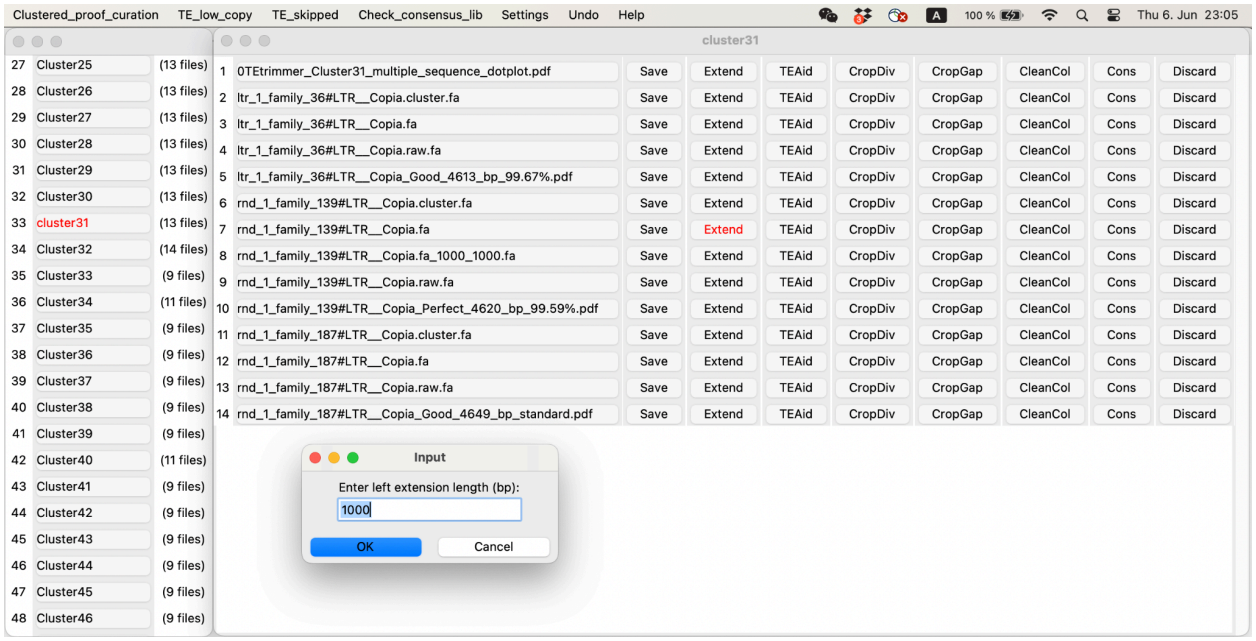
5. Additional proof curation (highly recommended)

You can use the TEtrimmer graphical user interface (GUI) tool to assist your proof curation. We highly recommend doing manual proof curation to increase the result quality.
TEtrimmer GUI can be used for Linux, macOS, and Windows. But “TEAid” function can’t be used for Windows systems.

To start TEtrimmer GUI tool:

```
python <path_to_TEtrimmer_output_directory>
/TEtrimmer_for_proof_curation/TEtrimmer_proof_anno_GUI/annoGUI.py -g
<genome.fa>
```


Clustered_proof_curation TE_low_copy TE_skipped Check_consensus_lib Settings Undo Help
TEtrimmer
Manual proof curation is highly recommended to improve the quality of TE annotations.
1, Click the <Clustered_proof_curation> button in the menu bar.
TEs with more than 90% identity are grouped into one cluster.
2, Click each <Cluster> button.
For each TE, you can find four files:
<seq_name.cluster.fa> multiple sequence alignment (MSA) before clustering
<seq_name.raw.fa> MSA file before cleaning
<seq_name.fa> MSA file corresponding to TE consensus sequence
<seq_name.pdf> report plots file for evaluating annotation quality
3, Double click <seq_name.pdf> to evaluate annotation quality.
4, If satisfied, click <Cons> button next to <seq_name.fa>.
This MSA file will be copied to <Proof_curation_consensus_folder>.
5, If not satisfied, check <seq_name.fa> or <seq_name.raw.fa>.
6, For more extension, click <Extend> button next to the fasta file.
Use <CropDiv> <CropGap> and <CleanCol> to clean the MSA.
Use <TEAid> to generate interactive report plots.
7, If still not satisfied, check <seq_name.cluster.fa>.
Select the sequences you want to use and click <Extension> to find the boundary
8, For skipped and low copy elements, evaluate by checking the PDF file.


Clustered_proof_curation TE_low_copy TE_skipped Check_consensus_lib Settings Undo Help
Thu 6. Jun 23:05
cluster31
27 Cluster25 (13 files)
28 Cluster26 (13 files)
29 Cluster27 (13 files)
30 Cluster28 (13 files)
31 Cluster29 (13 files)
32 Cluster30 (13 files)
33 cluster31 (13 files)
34 Cluster32 (14 files)
35 Cluster33 (9 files)
36 Cluster34 (11 files)
37 Cluster35 (9 files)
38 Cluster36 (9 files)
39 Cluster37 (9 files)
40 Cluster38 (9 files)
41 Cluster39 (9 files)
42 Cluster40 (11 files)
43 Cluster41 (9 files)
44 Cluster42 (9 files)
45 Cluster43 (9 files)
46 Cluster44 (9 files)
47 Cluster45 (9 files)
48 Cluster46 (9 files)
1 0TEtrimmer_Cluster31_multiple_sequence_dotplot.pdf
2 ltr_1_family_36#LTR__Copia.cluster.fa
3 ltr_1_family_36#LTR__Copia.fa
4 ltr_1_family_36#LTR__Copia.raw.fa
5 ltr_1_family_36#LTR__Copia_Good_4613_bp_99.67%.pdf
6 rnd_1_family_139#LTR__Copia.cluster.fa
7 rnd_1_family_139#LTR__Copia.fa
8 rnd_1_family_139#LTR__Copia.fa_1000_1000.fa
9 rnd_1_family_139#LTR__Copia.raw.fa
10 rnd_1_family_139#LTR__Copia_Perfect_4620_bp_99.59%.pdf
11 rnd_1_family_187#LTR__Copia.cluster.fa
12 rnd_1_family_187#LTR__Copia.fa
13 rnd_1_family_187#LTR__Copia.raw.fa
14 rnd_1_family_187#LTR__Copia_Good_4649_bp_standard.pdf
Save Extend TEAid CropDiv CropGap CleanCol Cons Discard
Input
Enter left extension length (bp):
1000
OK
Cancel

You can browse all the files by double-clicking the file name.
The MSA can be extended by clicking the “Extend” button.
The “TEAid” button is designed to generate interactive plots. Note: this is usable for macOS and Linux.
MAS cleaning functions are also integrated into the GUI named “CropDiv”, “CropGap”, and “CleanCol”.
“CropDiv” and “CropGap” can clean MSA rows.
“CleanCol” can remove MSA gappy columns.
You can modify MSA cleaning function parameters by “Settings” in the menu bar.

6. FAQs

Can I run TEtrimmer on a laptop or cluster?

TEtrimmer can be executed on both laptops and clusters. Refer to the examples in section 3.1.
Note: If your run stops for any reason, you can use the "--continue_analysis" option to resume from where you left off without the need to start over.

Which preset setting should I use for my organism? [--preset conserved or divergent?]

If you have little idea about how conserved TEs are in your organism, you can start with the setting of “divergent”. The “divergent” setting will probably take longer than the “conserved” setting but it might have better performance. Using the "divergent" setting on conserved TEs will not compromise the output performance; however, the opposite might be true.

7. Reference

Flynn, Jullien M., Robert Hubley, Clément Goubert, Jeb Rosen, Andrew G. Clark, Cédric Feschotte, and Arian F. Smit. 2020. “RepeatModeler2 for Automated Genomic Discovery of Transposable Element Families.” *Proceedings of the National Academy of Sciences* 117 (17): 9451–57.

Ou, Shujun, Weija Su, Yi Liao, Kapeel Chougule, Jireh RA Agda, Adam J. Hellinga, Carlos Santiago Blanco Lugo et al. "Benchmarking transposable element annotation methods for creation of a streamlined, comprehensive pipeline." *Genome biology* 20 (2019): 1-18.