trnanalysis Documentation

Release 0.0.1

Adam Cribbs

May 04, 2021

Contents:

This pipeline was developed to accurately map small RNA sequencing data and then perform accurate mapping of tRNA reads and qualitatively analyse the resulting data. trnanalysis has an emphasis on profiling nuclear and mitochondrial tRNA fragments.

CHAPTER 1

Citation

Currently we have a bioRxiv pre-print available:

tRNAnalysis: A flexible pre-processing and next-generation sequencing data analysis pipeline for transfer RNA

CHAPTER 2

Support

- Please refer to the FAQ section
- For bugs and issues, please raise and issue on _github

2.1 Indices and tables

2.1.1 Installation

The following sections describe how to install tRNAnalysis.

Conda Installation

The our preferred method of installation is using conda. If you don't have conda installed then please install conda using miniconda or anaconda.

We have been experiencing issues with installation because of channel priorities. Bioconda recommend that the channel priority for conda be set by running the following in the terminal:

```
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
```

tRNAnalysis is currently installed using the bioconda channel and the recipe can be found on **'github'_**.

To install tRNAnalysis:

```
conda create -n trnanalysis
conda activate trnanalysis
conda install -c bioconda trnanalysis
```

Pip installation

We recommend installation through conda because it manages the dependencies. However, tRNAnalysis can also be installed easily using the pip package manager. However, you will also have to install other dependencies manually:

```
pip install trnanalysis
```

Manual installation

To obtain the latest code, check it out from the public git repository and activate it:

```
git clone https://github.com/Acribbs/tRNAnalysis.git
cd tRNAnalysis
python setup.py install
```

Once checked-out, you can get the latest changes via pulling:

```
git pull origin master
```

Installing additional software

When building your own workflows we recommend using conda to install software into your environment where possible.

This can easily be performed by:

```
conda search <package>
conda install <package>
```

2.1.2 Cluster configuration

Our pipeline is developed using CGAT-core as the workflow engine. For more information on how tRNAnalysis is written and executed. In order for our workflows to be executed over a cluster you will need to configure the cluster options by following the example below:

Currently SGE, SLURM, Torque and PBSPro workload managers are supported. The default cluster options for cgatcore are set for SunGrid Engine (SGE). Therefore, if you would like to run an alternative workload manager then you will need to configure your settings for your cluster. In order to do this you will need to create a `.cgat.yml` within the user‘s home directory.

This will allow you to overide the default configurations. To view the hardcoded parameters for cgatcore please see the parameters.py file.

For an example of how to configure a PBSpro workload manager see this link to this config example.

The .cgat.yml is placed in your home directory and when a pipeline is executed it will automatically prioritise the `.cgat.yml` parameters over the cgatcore hard coded parameters. For example, adding the following to the .cgat.yml file will implement cluster settings for PBSpro:

```
      memory_resource: mem

options: -l walltime=00:10:00 -l select=1:ncpus=8:mem=1gb

queue_manager: pbspro
```

(continues on next page)

(continued from previous page)

```
queue: NONE

parallel_environment:  "dedicated"
```

2.1.3 Running a pipeline

Running tRNAnalysis is easy using the commandline. If you have installed trnanalysis using conda then all the software dependancies should have been installed and you are ready to go. A step by step tutorial pipeline can be found here:.

Introduction

This pipeline requires the following input:

- a single end fastq file - if you have paired end data we recommend flashing the reads together to make a single file or only using the first read of your paired end data.
- a bowtie indexed genome
- ensembl gtf: we recommend that you download our gtf files that have been sanitised for this workflow here. However,

you can use your own, if you make sure that all of the chromosomes are listed according to the ensembl annotations (i.e. the chromosomes are named chr1, chr2.. e.c.t.)

Optionally to make the pipeline run faster you can also use a downloaded tRNAscan-SE output. The most time consuming part of the pipeline is running tScan-SE to identify tRNAs across the genome. In order to speed the pipeline execution we have pre-ran tScan-SE and generated the outputs that can be found in the following directory . You can then tell the pipeline the location of the file using the yml configuration file.

Running tRNAnalysis

Command line usage information is available by running:

```
trnanalysis --help
```

The basic syntax for running tRNAnalysis is:

```
trnanalysis [workflow options] [workflow arguments]
```

`workflow options` can be one of the following:

make <task>

run all tasks required to build task

show <task>

show tasks required to build task without executing them

plot <task>

plot image of workflow (requires inkscape) of pipeline state for task

touch <task>

touch files without running task or its pre-requisites. This sets the timestamps for files in task and its pre-requisites such that they will seem up-to-date to the pipeline.

config

write a new configuration file `pipeline.ini` with default values. An existing configuration file will not be overwritten.

clone <srcdir>

clone a pipeline from `srcdir` into the current directory. Cloning attempts to conserve disk space by linking.

Fastq naming convention

tRNAanalysis assume that input fastq files follows the following naming convention(with the read inserted between the fastq and the gz). The reason for this is so that regular expressions do not have to account for the read within the name. It is also more explicit:

```
sample1-condition-R1.fastq.1.gz
sample1-condition-R2.fastq.2.gz
```

Additional options

In addition to running tRNAanalysis with default command line options, running trnaanalysis with –help will allow you to see additional options for `workflow arguments` when running the pipelines. These will modify the way the pipeline in ran.

- *-no-cluster*

This option allows the pipeline to run locally.

- *-input-validation*

This option will check the pipeline.ini file for missing values before the pipeline starts.

- *-debug*

Add debugging information to the console and not the logfile

- *-dry-run*

Perform a dry run of the pipeline (do not execute shell commands)

- *-exceptions*

Echo exceptions immediately as they occur.

-c - -checksums

Set the level of ruffus checksums.

Building tRNAnalysis reports

Reports are generated using the following command once a the *full* command has completed:

```
tranalysis make build_report
```

Troubleshooting

Many things can go wrong while running the pipeline. Look out for

- bad input format. The pipeline does not perform sanity checks on the input format. If the input is bad, you might see wrong or missing results or an error message.
- pipeline disruptions. Problems with the cluster, the file system or the controlling terminal might all cause the pipeline to abort.
- bugs. The pipeline makes many implicit assumptions about the input files and the programs it runs. If program versions change or inputs change, the pipeline might not be able to deal with it. The result will be wrong or missing results or an error message.

If tRNAnalysis aborts, locate the step that caused the error by reading the logfiles and the error messages on stderr (`nohup.out`). See if you can understand the error and guess the likely problem (new program versions, badly formatted input, . . .). If you are able to fix the error, remove the output files of the step in which the error occurred and restart the pipeline. Processing should resume at the appropriate point.

Note: Look out for upstream errors. For example, you may find that if the pipeline errors and stops, it may create the file and when the pipeline is started again, it will move to the next function, despite the previous file being empty. To fix this, delete the files created by the last task ran before restarting the pipeline.

Common errors

One of the most common errors when running the tRNAnalysis is:

```
GLOBAL_SESSION = drmaa.Session()
NameError: name 'drmaa' is not defined
```

This error occurs because you are not connected to the cluster. Alternatively you can run the pipeline in local mode by adding - *-no-cluster* as a command line option.

2.1.4 Running tRNAnalysis - Tutorial

Before beginning this tutorial make sure you have tRNAnalysis installed correctly, please see here (see *Installation*) for installation instructions.

In the following section we will run a toy example pipeline that demonstrates the functionality of tRNAnalysis. tRN-Analysis can be run locally or distributed across a cluster. This tutorial will explain the steps required to run tRNAnal-ysis.

Tutorial start

1. First download the tutorial data:

```
wget https://www.cgat.org/downloads/public/adam/trnanalysis/test_trna.tar.gz
tar -zxvf test_trna.tar.gz
```

2. Next we will generate a configuration yml file so the pipeline output can be modified:

```
cd test_trna
trnanalysis config
```

This will generate a **pipeline.yml** file containing the configuration parameters than can be used to modify the output of the pipeline. These parameters can all be modified to change the output or running of tRNAnalysis. However, for this tutorial you do not need to modify the parameters to run the pipeline as the default ones are appropriate in this instance.

Note: There is already a pipeline.yml file within the test data so you don't really need to run the command above this time.

3. Next we will run the pipeline:

```
trnanalysis make full -v5 --no-cluster
```

This `--no-cluster` will run the pipeline locally if you do not have access to a cluster. Alternatively if you have a cluster remove the `--no-cluster` option and the pipeline will distribute your jobs across the cluster. For information on how to configure your cluster please see the cluster config help.

Note: There are many commandline options available to run the pipeline. To see available options please run `trnanalysis --help`.

The pipeline is mostly quite quick to execute. However, it can take a considerable time to generate the bowtie indexes, in the future we will parameterise this so you can use pre-configured bowtie indexes so these do not have to keep being generated each time the pipeline is ran.

4. Generate a report

The final step is to generate a report to display the output of tRNAnalysis. The tutorial contains all of the files already set up for you to run the report without the need to modify any of them. However, we recommend that you look at the *The tRNAnalysis report* for how to set up the clustering and differential expression correctly.

The report is generated by running the command:

```
trnanalysis make build_report -v 5 --no-cluster
```

This will generate a MultiQC report in the folder *MultiQC_report.dir/* and an Rmarkdown report in *R_report.dir/Final_report/index.html*.

However, you can access the report by double clicking the FinalReport.html file within the directory that tRNAnalysis was executed.

This completes the tutorial for running the tRNAnalysis , hope you find it as useful as we do for analysing tRNA sequencing data.

2.1.5 The tRNAnalysis report

Before running the report

For the report to run successfully you will need to set up a design file (Actually, you can have multiple design files in there and all will run simultaneously) that controls how the differential expression model and contract will be performed.

The file has the naming convention design_<test>_<control>_<test>_<column>.csv

- <*test*> - refers to the test that you plan to run. There are two options "ltr" or "wald".
- <*control*> - This is the name of your control condition i.e. the samples you want to test against. This should match one of the samples within the <column> of the design file.

- <*test*> - This is the name of the test condition. This should match with one of the samples in the <column> of the design file
- <*column*> - This is a column name that you want to use for your testing in the design_* file

tRNAnalysis runs DEseq2 under the hood so you can refer to the bioconductor help pages for further information on how to set up contrasts for your data.

The design file should be laid out as follows, with the last model column detailing the model that you would like DESeq2 to run for the data.

Table 1: design layout

Sample	condition	sample_type	model
NORMAL_10KP_2481_miRNA	NO1	cell1	~condition
NORMAL_PLAC_2373_miRNA	plac1	cell1	
NORMAL_PLAC_2433_miRNA	plac1	cell2	

Opening the report

Once you have ran the software with the command:

```
trnanalysis make full
```

and:

```
trnanalysis make build_report
```

a final html report should have been generated.

In order to access this report please open the file called FinalReport.html which will be located within the directory that you ran tRNAnalysis. This should open the report in your preferred browser.

Understanding the report

The report contains a number of tabs at the top that display the analysis performed when the pipeline was ran. All of the report is annotated, which should make understanding the report output easier. However, if there are sections that you feel could be explained better then please raise an issue on github.

2.1.6 Developers

The following individuals are the main developers of the tRNAnalysis

Adam Cribbs

Anna James-Bott

2.1.7 Contributing

Contributions are very much encouraged and we greatly appreciate the time and effort people make to help maintain and support out tools. Every contribution helps, please dont be shy, we dont bite.

You can contribute to the development of our pipeline in a number of different ways:

Reporting bug fixes

Bugs are annoying and reporting them will help us to fix your issue.

Bugs can be reported using the issue section in github

When reporting issues, please include:

- Steps in your code/command that led to the bug so it can be reproduced.
- The error message from the log message.
- Any other helpful info, such as the system/cluster engine or version information.

Proposing a new feature/enhancement

If you wish to contribute a new feature to the pipeline then the best way is to raise this as an issue and label it as an enhancement in github

If you propose a new feature then please:

- Explain how your enhancement will work
- Describe as best as you can how you plan to implement this.
- If you dont think you have the necessary skills to implement this on your own then please say and we will try our best to help (or implement this for you). However, please be aware that this is a community developed software and our volunteers have other jobs. Therefore, we may not be able to work as fast as you hoped.

Pull Request Guidelines

Why not contribute to our project, its a great way of making the project better, your help is always welcome. We follow the fork/pull request model. To update our documentation, fix bugs or add extra enhancements you will need to create a pull request through github.

To create a pull request perform these steps:

1. Create a github account.
2. Create a personal fork of the project on github.
3. Clone the fork onto your local machine. Your remote repo on github is called `origin`.
4. Add the orginal repository as a remote called `upstream`.
5. If you made the fork a while ago then please make sure you `git pull upstream` to keep your repository up to date
6. Create a new branch to work on! We usually name our branches with capital first and last followed by a dash and something unique. For example: `git checkout -b AC-new_doc`.
7. Impliment your fix/enhancement and make sure your code is effectively documented.
8. Our code has tests and these will be ran when a pull request is submitted, however you can run our tests before you make the pull request, we have a number written in the `tests/` directory. For example: to run our import tests please run `nosetests tests/test_import.py`.
9. Add or change our documentation in the `docs/` directory.
10. Squash all of your commits into a single commit with gits interactive rebase.
11. Push your branch to your fork on github `git push origin`

12. From your fork in github.com, open a pull request in the correct branch.
13. … This is where someone will review your changes and modify them or approve them …
14. Once the pull request is approved and merged you can pull the changes from the `upstream` to your local repo and delete your branch.

Note: Always write your commit messages in the present tense. Your commit messages should describe what the commit does to the code and not what you did to the code.

2.1.8 FAQs

As our workflow develops we will add frequently asked questions here.

In the meantime please add issues to the github page

2.1.9 Licence

tRNAnalysis is an open-source project and we have made the repository available under the open source permissive free MIT software licence, allowing free and full use of the code for both commercial and non-commercial purposes. A copy of the licence is shown below:

MIT License

- genindex
- modindex
- search