Hunt Lab Small RNA Pipeline Docs

Kieran Reynolds

This tool is designed to automate a lot of the common bioinformatics performed on small RNA by the Hunt Lab. It is deliberately designed to be highly configurable so it can be easily applied to a wide range of scenarios.

Installation

You can obtain the code from https://github.com/kieranr51/HuntLab-smallRNA. To download, click the code button and copy the URL. You can then download it to a computer with git installed on it by opening the command line and running:

```
$ git clone <copied_url>
```

It is recommended to use conda to install this pipeline. To perform the installation, ensure conda is installed on the machine, navigate to the directory containing the code and `environment.yml` file, then run:

```
$ conda env create -f environment.yml
```

This will create a conda environment called `huntlab-smallrna`. This can then be activated each time you want to use the software with the command:

```
$ conda activate huntlab-smallrna
```

By default this provides the `hlsmallrna` executable. This provides all of the tasks listed below, each one can be run by typing `hlsmallrna` followed by the name and options for the task.

Tasks

Currently the following tasks have been implemented in it:

- **Process -** Trim adapters off raw RNA-Seq data, run a quality control report and trim of any bases that fall below a specified threshold from all the RNA

- **Sort** - Align the RNA against a given reference, removing any sequences that fail to align, then split them into files based on sequence length to allow for easy further processing. Can also remove sequences below a minimum length and above a maximum length if desired

- **ExtractNC** - Using a genome and GFF file containing annotations, extract the region that is transcribed, but not a coding region

- **Unitas** - Run a directory of RNA files (e.g. output of sort step) though unitas to classify the types of RNA, then combine the summaries into one spreadsheet for easy viewing and manipulation

- **TargetID -** Identify which of a set of potential targets a set of small RNA are targeting by reverse complimenting them and aligning them to the potential targets

More in depth documentation on each of these steps, starting with examples and suggestions for further processing. Further down there is a CLI Reference, that contains general help for each command, including what files are required and what outputs you get from each step.

Example Usage

To use this pipeline, you will need at least a FASTQ file containing raw small RNA-Seq reads. Then depending on what you want to achieve, you can run different steps of the pipeline. If you ever get stuck and need to see the options of a particular stage, all parts have their own help message, that can be seen by passing `--help` after the name of the stage.

Before starting though, you will need to create a configuration file that allows you to specify some parameters in advance, so you don't have to keep retyping them. By default, it looks for these in a file in your current directory called `config.toml,` though alternatives can be specified by passing the name to the `-C` argument before the name of the stage to run. More details on this can be found in the 'Configuration file' section below, but for now setup a simple file to specify an output directory of `output/` by putting the following in `config.toml:`

```
[general]
output_directory = "./output"
```

In addition, check that all of the commands can be run by just typing their name in the command line, as this is how the program tries to use them by default. If they can't, you can set a custom path by setting the relevant `path_to_` variable in the configuration file. For example if unitas is run with `unitas.pl` instead of just `unitas` add the following to the bottom of the config file:

```
[cli-tools]
[cli-tools.unitas]
path_to_unitas = "unitas.pl"
```

Process

The first stage is Process. This is for if you have raw data that hasn't had it's adapters trimmed and been quality filtered, so if your data has had this done to it already, this step can be skipped. To run you need a FASTQ file containing raw small RNA and the sequence and end of any adapters attached during sequencing.

Once you have this data, to include the trimming in the run, you need to set one of three flags with the adapter sequence, depending on the position of the particular adapter.

If your adapter is on the 3' end only, use `-a;` if it is from the 5' end only, use `-g` and if it could be on either end use `-b.` If you don't need adapters trimmed, just don't specify any of them and the step will be skipped. For example, if your small RNAs are in `smallRNA.fastq` and you want to trim the adapter sequence `AGCATA` from the 3' end only, you would run:

```
$ hlsmallrna process -a AGCATA smallRNA.fastq
```

In addition, this step runs FastQC to produce a report on the quality of the small RNA and automatically cuts anything with a lower quartile of quality below a cutoff. By default, this is set to 20, but can be changed by the user using the `-c` flag. If you do not want this, you can set `-c 0` and the FastQC step will be skipped for efficiency. For example if you want to change the cutoff for the last command to 5, you could run:

```
$ hlsmallrna process -c 5 -a AGCATA smallRNA.fastq
```

This produces a number of files in the output directory, importantly `cut_sequences.fastq` has the sequences that have adapters trimmed and low quality parts removed. (See CLI Reference section for information on other output for all of the commands)

Sort

The second stage is Sort. This stage aligns the small RNAs to the genome of the organism and filters out any that don't align, to remove contamination. Then it splits the remaining reads into FASTQ files by length, to allow for easier classification.

To run this, you need a FASTQ file of processed small RNA and a FASTA file containing the genome of the organism of interest. For example, if your small RNA are in `smallRNA.fastq` and your genome is in `genome.fasta` you can run:

```
$ hlsmallrna sort smallRNA.fastq genome.fasta
```

If you are only interested in a subset of lengths of small RNA, this step can be set to restrict the lengths it outputs with `-l` for minimum and `-x` for maximum. For example, if you only want small RNA with lengths between 12 and 30, you could run:

```
$ hlsmallrna sort -l 12 -x 30 smallRNA.fastq genome.fasta
```

By default this aligns the small RNA to the genome with bowtie2's default settings, which allow different amounts of mismatches depending on how long the RNA is. Details of this can be found in the score configuration options in the manual, that can be found at: https://bowtie-bio.sourceforge.net/bowtie2/manual.shtml. To deal with cases where we may want to do a more stringent run, the `-m` or `--ref-mismatches` argument can be supplied with the number of mismatches for bowtie2 to allow. For example, if you want to perform an operation using the same files as above, but only include small RNA that map exactly to the genome, you could run:

```
$ hlsmallrna sort -m 0 smallRNA.fastq genome.fasta
```

This command produces a few useful files in the output directory:

- `binned_rna/` - directory containing one fastq file for each length of small RNA
- `rna_length_report.csv` – table showing a summary of the RNAs by length and first base
- `baseplot.png` - plot of the length and first base of the RNAs (this is sometimes wrong, replot from the csv values for now, will fix matplotlib code soon)

Unitas

The third stage is Unitas. This runs unitas on a directory containing a range of lengths of small RNA to classify the origin of the RNA. To do this, you will need files containing reference sets of the genome targets you wish to find, with labels in their IDs and a directory containing small RNA of different lengths, similar to the output of sort.

You can set the reference file using the `-r` command line argument, but I think it is easier to set them in the configuration file, as it makes the command shorter to type. If your reference files are in the same directory you are running in and called `ref1.fasta, ref2.fasta` and `ref3.fasta`, you can add the following to the config file:

```
[command]
refseq = [
    "ref1.fasta",
    "ref2.fasta",
    "ref3.fasta"
]
```

Note that all of these reference sequences should be labelled with the name of the category they belong to and the | (bar or pipe character) before the start of their ID. If this hasn't been done and all of the sequences in a FASTA file should be labelled the same, you can use the `label_for_unitas` script that was installed with the small RNA pipeline. For example if you want to label the sequences in `unlabelledTEs.fasta` with the label TE, with the result produced in `teRef.fasta`, you could run:

```
$ label_for_unitas "TE" unlabelledTEs.fasta -o teRef.fasta
```

Once the correct reference file are set, if your small RNA, split by length, are in `output/binned_rna,` you can run:

```
$  hlsmallrna unitas output/binned_rna
```

This produces a useful summary and graph in the output directory:

- `unitasGraph.png -` graph showing the categories allocated by unitas against the length of the small RNA (this is sometimes wrong, replot from the csv values for now, will fix matplotlib code soon)

- `unitas_graph_data.csv -` summary of the raw numbers produced from the unitas runs, used to produce the graph

If you want to get small RNA that are derived from genes, you will need to obtain both of the mRNA and CDS regions and pass them to speciled arguments in the pipeline so they can be combined. This can either be done using the config file as followers or with the corresponding command line arguments (--cds and --unspliced-transcriptome):

```
[command]
refseq = [
    "ref1.fasta",
    "ref2.fasta",
    "ref3.fasta"
]
cds = “cds_sequences.fasta”
unspliced_transcriptome = “unspliced_transcriptome.fasta”
```

Extract Non-Coding

Sometimes you want to see if a small RNA comes from a non-coding region of the transcriptome, so the pipeline contains a utility command to automatically extract this using a genome and some annotations. At minimum, these annotations need to contain a transcribed region labelled with ‘mRNA’ and a coding region within the transcribed region labelled ‘CDS’. Most of the ones on Wormbase Parasite are fine for this purpose.

As an example, to extract the regions from `genome.fasta` using the annotations in `annotations.gff` you could run:

```
$ fungi
```

This will produce a FASTA file containing the non-coding regions, complete with unitas labels applied, called `noncoding.fasta` in the output directory.

Target Identification

The fourth and final main step is Target Identification. This takes a set of small RNA and a set of potential targets and produces a list of genome features that are targeted by each small RNA.

As with unitas, you can set target files using the command line option `-t` but it is easier to add them to the configuration file. For example if your target files are `target1.fasta`, `target2.fasta` and `target3.fasta`, they can be set by adding the following in the section below `[command]`:

```
target_files = [
    "target1.fasta",
    "target2.fasta",
    "target3.fasta"
]
```

Then, if your small RNA are in `smallRNA.fastq` you can run the following:

```
$ hlsmallrna targetid smallRNA.fastq
```

Make sure the sequences in both `smallRNA.fastq` and each of the reference files have unique sequence ids, otherwise the command will fail.

If you are working with very short sequences (<5 bases), you will need to reduce the minimum length with the `-m` option. Conversely, if you know your minimum length is longer than this, this step can be sped up by increasing it. For example, if you know the shortest sequence you are dealing with is 10 bases long, you could run:

```
$ hlsmallrna targetid -m 10 smallRNA.fastq
```

Though this is not necessary and the program will produce a correct result for sequences longer than 5 without it, so only add if you need the extra speed.

This produces `rna_target_list.csv` which contains a list of small RNA target pairs and some general information about each pair’s alignment.

If you are dealing with a small RNA that doesn’t need perfect complementarity, you can add the `--num-mismatches` option to set the number of mismatches the alignment should allow. For example, if you want to allow up to four mismatches, you could run:

```
$ hlsmallrna targetid --num-mismatches 4 smallRNA.fastq
```

All

Since the first three steps are often run one after the other, a shorthand command `all` is provided for convenience. This takes all of the arguments from the Process, Sort and Unitas steps, then runs them, piping the result from one into the next. All files produced by those steps are then produced in the output directory.

For example if you have small RNAs in `smallRNA.fastq,` a genome in `genome.fasta` and `refseq` set in the config file and you want to trim the adapter sequence `ACATA` from both ends, but not run quality filtering you could run:

```
$ hlsmallrna all -c 0 -b ACATA smallRNA.fastq genome.fasta
```

Plotting Genome Location

Basic Plot

A common task you might want to do is plotting the location of things within the genome, for example, genome features you are interested in or the positions of small RNAs you have aligned to the genome. To make this easy, a script is included in this environment called `build_coord_files`. This automatically generates tsv files for use with the R package `chromPlot`. This section explains how you can generate a plot with it.

Firstly, you need at least two files: a full genome in a FASTA or FASTQ file and a set of locations you are interested in, either as an alignment output (in SAM or BAM format) or as feature annotations in GFF format. To start produce the genome coordinate file with the command: (assuming `genome.fasta` is your genome file)

```
$ build_coord_files -o genome_coord.tsv genome.fasta
```

By default, the script will figure out what type of output to produce by the file extension. But if it can't, the user will need to provide a flag to tell it what type of input you are using (in this case `--fasta`). Next we produce the coordinates file for the item of interest, assuming it is in `alignment.sam` you can run:

```
$ build_coord_files -o alignment_coord.tsv alignment.sam
```

This will produce two files, one for coordinates on the sense strand and one for coordinates on the antisense strand. Finally, we can use these files to plot the result. If it isn't already, install `chromPlot` with bioconducter using the following R command:

```
> BiocManager::install("chromPlot")
```

Then you should be able to create a plot using the following R code (of two example chromosomes called ‘SRAE_chr1’ and ‘SRAE_chr2’):

```
chrom_coords <- read.csv("/path/to/genome_coord.tsv", sep = "\t")
sense_pos <- read.csv("/path/to/sense_alignment_coord.tsv", sep = "\t")
antisense_pos <- read.csv("/path/to/antisense_alignment_coord.tsv", sep = "\t")

library(chromPlot)

chromPlot(
  gaps = chrom_coords, annot1 = sense_pos, annot2 = antisense_pos, plotRndchr = T,
  chr = c("SRAE_chr1", "SRAE_chr2"), bin = 50000,
  chrSide = c(-1, 1, -1, -1, 1, -1, -1, 1)
)
```

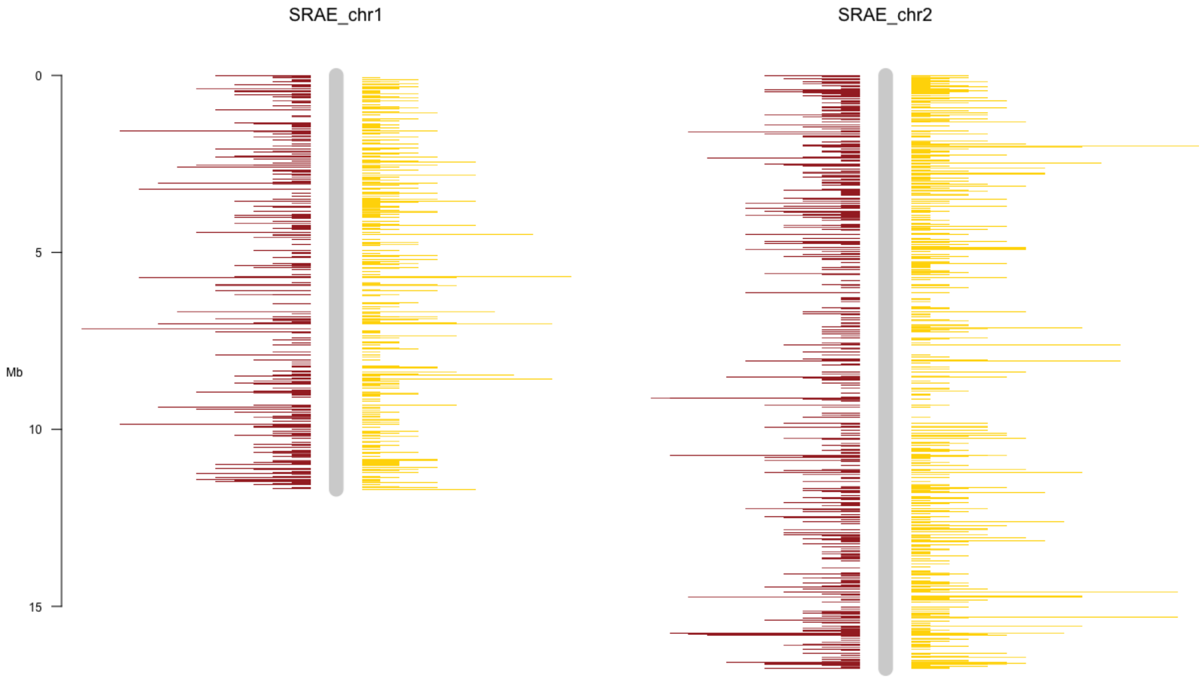
The script breaks down as follows:

- First three lines load in the data we just produced with `build_coord_files`

- Following line loads the `chromPlot` library into R

- The final line plots the data with `chromPlot`:
    - `gaps` defines what each chromosome should look like
    - `annot1` and `annot2` define the annotation data to show
    - `plotRndchr` disables the assumption in `chromPlot` that the chromosomes are labelled with numbers
    - `chr` selects which chromosomes to plot by specifying an array of names
    - `bin` sets the bin size to group the features into when plotting
    - `chrSide` defines which side of the chromosome to plot each feature, the only change we have made from the default is setting the second item to 1 as opposed to -1. This ensures `annot2` is plotted on the opposite side to `annot1`

The resulting plot will look something like:



Dealing with Multiple Scaffolds

In more complex cases, one or more of your chromosomes may be split into multiple scaffolds. `Build_coord_files` has a scaffold aware mode that can automatically try to merge these, but it will only work if your scaffold's IDs are in the form of `<chromosome_name>_scaffold<scaffold_number>` e.g. `SRAE_chrX_scaffold1`. Once you have this, you can generate the data files with the `-c` flag set, as follows:

```
$ build_coord_files -c -o genome_coord.tsv genome.fasta
$ build_coord_files -c --genome-coords genome_coord.tsv -o
alignment_coord.tsv alignment.sam
```

Note you need to provide the genome coords file when generating files for what you are plotting, this allows for a correction of coordinates to be made from where scaffolds are merged. In addition to the regular files, this produces `scaffold_info_genome_coord.tsv`, that can be used to add scaffold dividers to your plots. To do this, the following R script can be run:

```
chrom_coords <- read.csv("/path/to/genome_coord.tsv", sep = "\t")
sense_pos <- read.csv("/path/to/sense_alignment_coord.tsv", sep = "\t")
antisense_pos <- read.csv("/path/to/antisense_alignment_coord.tsv", sep = "\t")
scaffolds <- read.csv("/path/to/scaffold_info_genome_coord.tsv", sep = "\t")
```

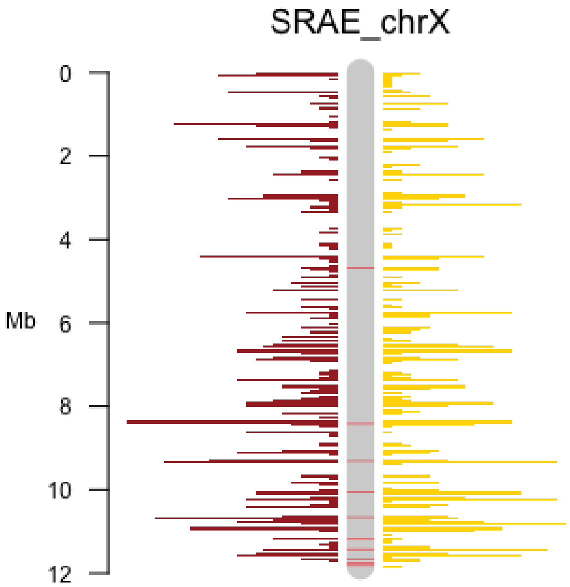
```
library(chromPlot)

chromPlot(
  gaps = chrom_coords, annot1 = sense_pos, annot2 = antisense_pos, plotRndchr = T,
  chr = c("SRAE_chr1", "SRAE_chr2"), bin = 50000, bands = scaffolds,
  chrSide = c(-1, 1, -1, -1, 1, -1, -1, 1)
)
```

Compared to the last script, this adds:

- The fourth line, that loads in the scaffold data
- The `bands` argument, that draws red lines at the end of each scaffold

Note that, by default the bands are 20,000 bp wide, as that looks good on the plot. At that size they may not appear if you make the bins too much bigger. Therefore, if you plan to make the bins bigger, pass a larger value to the `--band-width` flag when creating the genome file. That makes the bands bigger, which results in a more visible line on the plot.

A plot produced by this method will look something like:



Further Processing Options

Signature Identification

AT/GC Richness

Nucleotide richness and the presence of any conserved motifs is identified using WebLogo (weblogo.threeplusone.com). Sequences need to be the same length and can be either uploaded using a fasta file or can be pasted in.

Upstream motif (e.g. piRNA)

map to genome with bowtie2, convert sam to bam with samtools, convert bamToBed with bedtools

```
$ bowtie2 -x genome.fasta -f -U sequences.fasta -S
sequences_mapped.sam
$ samtools view sequences_mapped.sam -o sequences_mapped.bam
$ bamToBed -i sequences.bam > sequences.bed
```

extract flanking sequences with flankBed in bedtools (need to use bedtools format genome file - generate with faidx and cut)

```
$ samtools faidx genome.fasta
$ cut -f 1,2 genome.fa.fai > chrom.sizes
$ flankBed -i sequences.bed -g chrom.sizes -b 100 >
sequences_100nt_flank.bed
```

or 60 upstream only

```
$ flankBed -i sequences.bed -g chrom.sizes -l 60 -r 0 >
sequnences_60nt.bed
```

extract fasta sequences from bed fastFromBed with bedtools

```
$ fastaFromBed -fi genome.fasta -bed sequences_100nt_Flank.bed -fo
sequences_100nt_Flank_seqs.fasta
```

Ping Pong signature

To find ping-pong signature, Unitas with the option -pp can be used to find 5’ overlaps of mapped sequence reads to the genome and calculate a Z-score for the enrichment of 10 bp overlaps.

```
$ perl unitas.pl -pp -input map.file (SAM or ELAND3 format) −species
x −refseq reference.fasta
```

Dicer Signature (StepRNA)

To install stepRNA, see the documentation on the GitHub page (https://github.com/bmm514/stepRNA).

stepRNA is able to identify a Dicer processing signature from a small RNA sequence dataset. It requires two FASTA files as input that have had their adapters already trimmed from the ends.

```
$ stepRNA -r/--reference REFERENCE.fa -q/--reads READS.fa
```

- `READS.fa` should contain all of the small RNA sequencing reads.
- `REFERENCE.fa` is usually filtered to investigate small RNAs of interest e.g. 26G sRNAs.

Identical reads in the READ and REFERENCE files can also be removed before searching for a Dicer signature, if desired, by using `-e/--remove_exact`.

Importantly the FASTA headers must be unique - this can be done by stepRNA with `-u/--make_unique`.

stepRNA will then generate:

- BAM alignment files (with different combinations of overhang lengths)
- Overhang length CSV
- Passenger Read Length CSV

These can then be used to plot the distribution and/or further explore the reads that have aligned.

Note: If the sRNA lengths in the read file are all the same i.e. 21 nt long, stepRNA will not be able to find overhangs due to the scoring system. In this case the `-m/--min_score` should be set to an appropriate value; we recommend sRNA length - 7nt (e.g. 21 - 6 = 15).

`$ stepRNA --help` to bring up the command line help, a description of the methods can be found in the stepRNA publication (add_when_available)

```
stepRNA [-h] -r REFERENCE -q READS [-n NAME] [-d DIRECTORY]
               [-m MIN_SCORE] [-e] [-u] [-j] [-V]

Align a reference RNA file to read sequences. Output will be a set of CSV files containing
information about the length of the reads, number of reads aligned to a reference sequence and
the length of overhangs of the alignment.
Reference RNA file will be automatically indexed

Optional Arguments:
  -h, --help            show this help message and exit
  -n NAME, --name NAME  Prefix for the output files
  -d DIRECTORY, --directory DIRECTORY
                        Directory to store the output files
  -m MIN_SCORE, --min_score MIN_SCORE
                        Minimum score to accept, default is the shortest read
                        length

Required Arguments:
  -r REFERENCE, --reference REFERENCE
                        Path to the reference sequences
  -q READS, --reads READS
                        Path to the read sequences

Flags:
  -e, --remove_exact    Remove exact read matches to the reference sequence
  -u, --make_unique     Make FASTA headers unique in reference and reads i.e.
                        >Read_1 >Read_2
  -j, --write_json      Write count dictionaries to a JSON file
  -V, --version         Print version number then exit.
```

Same Strand Overlap

If you wish to detect same strand overlaps, you can use the `overlap_ss` script that is installed in the same environment as this pipeline. This uses a number of bash programs to create files that show where two sets of small RNA that appear on the same strand overlap.

To use this script you need two sets of small RNA as FASTA or FASTQ files, one for the base and one for the overlap, plus the genome of the species of interest. You can then run the script in the following way:

```
$ overlap_ss genome.fasta smallRNA1.fastq smallRNA2.fastq
```

This script will then create output in the output directory specified in `config.toml` under the directory `samestrand_overlap/.`

miRNA

Notes on Settings

Configuration file

For ease of use this program requires a configuration file in TOML format to run successfully. At minimum it can be a blank file, but I strongly suggest setting at least the `output_directory` setting to be named something more useful than `output,` the default setting.

By default the program tries to use `config.toml` in your current directory, but it can be set to a different file using the `-C` option before specifying which task to run, e.g.:

```
$ hlsmallrna -C alternate.toml sort ...
```

The layout of a TOML file has a number of sections each with a number of configuration keys. Each key has an expected type (e.g. string, integer, floating point number etc.) and the program will check they are as expected before running. If one is wrong, the program will crash and alert the user of the problem. A list of these keys and example values can be found in the `README.md` included with the program.

Manually setting parameters of internal commands

For advanced uses, you may want to add parameters to the internal commands run. This can be done through variables in the `cli-tools` section of the configuration file. A full list of these parameters can be found in the `README.md` provided with the code.

There is a main one that you can set, called `<name>_params`. It is a list of custom parameters to be added on to the command when called. For example if you wanted to set min-length to 10 for FastQC you could add the following to the configuration file:

```
[cli-tools]
[cli-tools.fastqc]
fastqc_params = ["--min_length", "10"]
```

Suppressing excess output

If you do not want to view the output from each of the commands, you can pass the `-q` option to the main script. (Before the task selection, like with `-C`). It is recommended to test the command first however, as this option sometimes suppresses useful error messages.

Note this doesn’t stop all output, only that of internal commands. If you don’t want any progress messages whatsoever, only errors and warnings from python, set `-q` twice. E.g. `-q -q` or `-qq.`

Troubleshooting

If it is taking too long to run or producing odd data, I strongly suggest deleting or moving the output directory somewhere else, then doing a clean run. This is because some steps use the data in the output directory as input, so may be combining old and new data.

When using the targetid command, samtools is used to filter out duplicates and create the FASTQ file outputs. Samtools is known to be picky about how the input file is layed out, so will fail. The program should keep running, but if you want these files, check the following in your input files:

- For FASTA files, there is no @ at the start of any of the header strings
- There are no duplicate headers in any of the input files

CLI Reference

References of how to use the command line interface of the program and notes on some ways particular commands work that could trip you up.

General help message is as follows:

```
usage: hlsmallrna [-h] [-q] [-C PATH_TO_CONFIG]
               {process,sort,extractnc,unitas,targetid,all} ...

Pipeline to process small RNAs

positional arguments:
  {process,sort,extractnc,unitas,targetid,all}
        process                Preprocessing for the RNA
        sort                   Find RNAs that align to a genome and sort them by
                       length
        extractnc              Extract the noncoding region from a fasta with a GFF
                       file
        unitas                 Run unitas on split files and merge results
        targetid               Align small RNA to a number of genome features to find
                       out what is targeted
        all                    Run process, sort and unitas one after the other

optional arguments:
  -h, --help           show this help message and exit
  -q, --quiet          Suppress output from intermediate commands
  -C PATH_TO_CONFIG, --path-to-config PATH_TO_CONFIG
                       Path to the TOML format config file to use
```

Process

Docs:

```
process [-h] [-a ADAPTER] [-g FRONT] [-b ANYWHERE]
                               [-c CUTOFF]
                               small_rna

positional arguments:
  small_rna            Path to FASTQ containing the small RNA

optional arguments:
  -h, --help           show this help message and exit
  -a ADAPTER, --adapter ADAPTER
                       Sequence of the adapter to remove from the 3' end
  -g FRONT, --front FRONT
                       Sequence of the adapter to remove from the 5' end
  -b ANYWHERE, --anywhere ANYWHERE
                       Sequence of the adapters to remove from both ends
  -c CUTOFF, --cutoff CUTOFF
                       Quality cutoff to trim RNA sequences at
```

Input files:

`Small_rna` - fastq file containing raw RNA-seq data

Output files:

`Fastqc/` - directory containing the raw output of FastQC

`Trimmed_rna.fastq -` file containing the raw RNA with adapters removed (only produced if an adapter sequence is provided)

`Cut_sequences.fastq -` sequences with low quality parts removed

Note that the parts of this step can easily be skipped. If you want to skip adapter trimming, don't specify any adapters with `-a, -g` or `-c.` If you don't want to run fastQC set `-c 0`. The program knows that in these cases, the steps do not have to be run.

Sort

```
sort [-h] [-d CDS] [-l MIN_LENGTH] [-x MAX_LENGTH]
                       [-m REF_MISMATCHES] [--disable-alignment]
                       small_rna [genome]

positional arguments:
  small_rna            Path to FASTQ containing the small RNA
  genome               Genome to align against

optional arguments:
  -h, --help           show this help message and exit
  -d CDS, --cds CDS    Optional CDS region, also align this to the CDS region
                       as well as the genome
  -l MIN_LENGTH, --min-length MIN_LENGTH
                       Minimum length to bin
  -x MAX_LENGTH, --max-length MAX_LENGTH
                       Maximum length to bin
  -m REF_MISMATCHES, --ref-mismatches REF_MISMATCHES
                       Number of mismatches to use in bowtie2, None for
                       default behaviour
  --disable-alignment   Skip the alignment to the reference genome step
```

Input files:

`Small_rna -` fastq file containing RNA with adapters removed

`Genome -` Reference genome of the species you are using to align against in fasta format

Output files:

`Bbmap_index/ -` contains the index of the reference genome created by bbmap

`Mapped_sequences.fastq -` sequences successfully mapped to the reference by bbmap `binned_rna/` - directory containing one fastq file for each length of sequence contained in the bbmap output

`Rna_length_report.csv` - table showing a summary of the RNAs by length and first base

`Baseplot.png` - plot of the length and first base of the RNAs, using the data in rna_length_report.csv

`Baseplot_data.csv` - raw data used to make baseplot.png to allow for easy regraphing

ExtractNC

Docs:

```
extarctnc [-h] genome gff_file

positional arguments:
  genome      FASTA containing the genome to extract from
  gff_file    GFF file containing annotations of CDS and mRNA regions

optional arguments:
  -h, --help  show this help message and exit
```

Input files:

`Genome` - file containing the whole genome of the species of interest

`Gff_file` - GFF3 file containing annotations at least for the mRNA and coding region (labelled CDS) for the genome

Output files:

`Noncoding.fasta` - FASTA file containing only the transcribed noncoding regions of the DNA

Note the method used here will not work if there a coding region is labelled outside a single mRNA region. The program does a check before running to make sure this is true and throws an exception before trying to run if so, to prevent nonsense results from being produced.

Unitas

Docs:

```
unitas [-h] [-d CDS] [-u UNSPLICED_TRANSCRIPTOME]
                      [-r [REFSEQ [REFSEQ ...]]] [-s SPECIES]
                      path_to_rnas

positional arguments:
  path_to_rnas        Path to the folder with varying length RNAs in
```

```
optional arguments:
  -h, --help            show this help message and exit
  -d CDS, --cds CDS     Optional CDS region, passed to unitas
  -u UNSPLICED_TRANSCRIPTOME, --unspliced-transcriptome UNSPLICED_TRANSCRIPTOME
                        Optional, unspliced transcriptome, passed to unitas
  -r [REFSEQ [REFSEQ ...]], --refseq [REFSEQ [REFSEQ ...]]
                        References for use with unitas
  -s SPECIES, --species SPECIES
                        Species to set in unitas arguments
```

Input files:

`Path_to_rnas -` path to the directory containing fastq files of each set of small RNAs you want to procesus in unitas

Note: to get unitas to work, you need to set either the species to a compatible species or provide one or more files of reference small RNAs. These can either be done with the `-s` and `-r` optional arguments or by adding them to your configuration file (see below).

`CDS -` Coding sequence, pipeline automatically labels with Gene and combines with the unspliced transcriptome, optional

`Unspliced_transcriptome -` Unspliced transcriptome, pipeline automatically labels with Gene and combines with the CDS, optional

Output files:

`unitas/ -` directory containing the results for all the unitas runs performed by the program

`Unitas_summery.csv -` CSV file containing the combined summaries of all of the unitas runs (plus some extra numbers). It is recommended to open it in a spreadsheet package (e.g. microsoft excel, libreoffice calc) as there is a lot of data and it may look a mess in a text editor.

`UnitasGraph.png -` graph showing the categories allocated by unitas against the length of the small RNA

`Unitas_graph_data.csv -` raw data used for creating unitasGraph.png to allow for easy regraphing

TargetID

Docs:

```
targetid [-h] [-m MIN_SEQ_LENGTH] [-t TARGET_FILES [TARGET_FILES ...]]
                              [--num-mismatches NUM_MISMATCHES]
                              small_rna


positional arguments:
  small_rna           Path to the FASTQ containing the small RNA to find targets of

optional arguments:
  -h, --help          show this help message and exit
  -m MIN_SEQ_LENGTH, --min-seq-length MIN_SEQ_LENGTH
                      Minimum sequence length to properly align
  -t TARGET_FILES [TARGET_FILES ...], --target-files TARGET_FILES [TARGET_FILES ...]
                      Files containing genome features that could be targeted
  --num-mismatches NUM_MISMATCHES
                      Number of mismatches to allow in the alignment, defaults to 0
```

Input:

`small_rna` - fastq file containing small RNA to look for targets of

`target_files` - one or more fastq files containing a list of potential target sequences

Output:

`Bowtie_indexes/` - indexes produced of the target files with bowtie2

`target_alignments/` - SAM files containing the results of alignment attempts and FASTQ files containing all the small RNA successfully aligned against the targets. One set of files are produced for each target file provided

`rna_target_list.csv` - list of RNA target pairs

Note that the `samtools view` command used in this step is the only one that has defaults pre-set in the `samtools_view_params` config key, namly `-h -F 256 -F 4`. They will be removed if you set the value differently in the config file. To only add parameters, you will need to set these ones again before the extra ones you want to set. E.g. `samtools_view_params = ["-h", "-F", "256", "-F", "4", ...]`

Make sure all of your target files are labelled with what they are, by prepending the name and | (the bar or pipe character) to the ID in the fasta file. If you don't do this, it will not be nicely sorted in the results correctly.

Label for Unitas

```
usage: label_for_unitas [-h] [-o OUTPUT] label file_path

Program to automatically prepend unitas labels to all the sequences in a FASTA
file

positional arguments:
  label               Label to prepend to the sequences in a fasta file
  file_path           Path to the fasta file to label

optional arguments:
  -h, --help           show this help message and exit
  -o OUTPUT, --output OUTPUT
                       File to output to, defaults to labelled.fasta
```

Build Coord Files

```
usage: build_coord_files [-h] [-a] [-q] [-s] [-b] [-g] [-r] [-c]
                         [--feature FEATURE] [--genome-coords GENOME_COORDS]
                         [--band-width BAND_WIDTH] [-o OUTPUT]
                         input_file

Convert common bioinformatics file formats into coordinate file to plot

positional arguments:
  input_file           File to convert, attempts to autodetect type

optional arguments:
  -h, --help           show this help message and exit
  -a, --fasta          Treat input as a FASTA file
  -q, --fastq          Treat input as a FASTQ file
  -s, --sam            Treat input as a SAM file
  -b, --bam            Treat input as a BAM file
  -g, --gff            Treat input as a GFF file
  -r, --rm-fa-out      Treat input as a RepeatMasker .fa.out file
  -c, --scaffold-aware  Merge scaffolds into one chromosome
  --feature FEATURE    Select a gff feature to use
  --genome-coords GENOME_COORDS
                       File containing the coordinates of the genome
  --band-width BAND_WIDTH
                       Size of the bands showing change in scaffolds
  -o OUTPUT, --output OUTPUT
                       File to output to
```

Overlap SS

```
usage: overlap_ss [-h] [-q] genome rna_file_1 rna_file_2

Looks for overlaps of two classes of RNA on the same strand of DNA

positional arguments:
  genome       FASTA file containing the genome of the organism
  rna_file_1   File containing the RNA to use as a base
  rna_file_2   File containing the RNA to look for overlaps with

optional arguments:
  -h, --help   show this help message and exit
  -q, --quiet  Suppress output from intermediate commands
```