Uniform Transcriptomics Pipeline

The Manual

Alejandro Baars
alebaars_98@outlook.com
Made as part of the profile research project
Utrecht University, December 2023

Table of Contents

1. Installation & setup

The pipeline serves as a “wrapper” set of scripts to use other programmes in a chained fashion. The scripts and programmes involved need to be installed manually. This first chapter will aid with that process. Make sure that your Linux system has Python (v3.6 or higher), R (v4.3 or higher), and Java (v1.8 or higher) installed. If this is not the case, they can also be included in Anaconda (see below).

1.1. Installing the required software

This pipeline uses various programmes to do most of the heavy lifting, such as the cleaning of your data and performing the alignment. The easiest way to get these programmes is using conda, a large, open-source repository which enables installation of programmes with a single command. It is accompanied with an installation guide. For biological programmes, it is necessary to include the Bioconda channel. Once these are set-up, you can use conda to create an environment, which is a sort of profile that has the desired software installed and can be activated and deactivated at will. A handy cheat-sheet is provided to get you started on this. A list of programmes can be found below, as well as how to use them and how to install them if conda is not an option. For all programmes, the newest available version is recommended.

TIP: You can navitage the Linux system using `cd` (change directory), `ls` (lists items), and `mkdir` (make directory).

TIP: You can download things directly onto a linux machine using the `wget` command.

FastQC

FastQC is a programme which measures the sequencing quality of your reads. It will give you several quality measures, such as the phred scores, GC or N content, sequence length distribution, etc. You can download the tool here. Documentation and some sample reports as html documents can be found here.

MultiQC

MultiQC is a collection of python tools to summarize many types of output typically found in bioinformatics. In this pipeline, it is used to summarize the FastQC reports, Picard CollectAlignmentSummaryMetrics reports, and Subread featureCounts reports, but it supports many more – 128 in total at the time of writing. You can find the necessary documentation here, and you can find installation instructions here.

Trimmomatic

Trimmomatic is a tool to trim your sequencing reads of adapters and low-quality sequences. Installation of this tool is not necessary. You can find how to install and use it on the github page. Keep in mind that, if you do not install it through conda, the programme is used directly from the .jar file; `java -jar path/to/Trimmomatic-x.xx.jar <args>`. If you are using a conda environment, the programme will have a “direct” command, along the lines of `Trimmomatic-x.xx.jar <args>`.

gffread

gffread is a tool to parse, test, and convert annotation files. In this pipeline, it is used to convert the provided annotation. This ensures that all tools can use the annotation accordingly. Instructions for usage and installation, as well as a pre-compiled library (which can simply be downloaded and extracted on the machine) can be found here.

STAR

STAR is a RNAseq alignment tool, which will take care of aligning the reads to the reference genomes. Links and instructions for manual installation are found in a PDF on their github. If you are working on a computer on which you have no admin rights, make sure you (try to) use the pre-compiled archives that are provided in chapter 1.1. More information can be found below in “During/after manual installation”.

Samtools

Samtools is a cornerstone programme in almost all bioinformatics to do with DNA and RNA. It can be used to analyze, format, and pipe alignment data in many ways. In this pipeline, it is used to separate the merged data per their respective genome. It can be downloaded from here and an overview of all available tools can be found here.

Picard

Picard is another collection of tools that plays an important role in analyzing and filtering alignment data. In this project, we only use to for quality control and to remove duplicate mappings. Instructions on downloading and using the programme can be found on its github page. Everything is compiled into a single .jar file, so no installation is required.

Subread (featureCounts)

Subread is a collection of tools for aligning sequencing reads, counting reads, and discovering variants. However, we will only be using the count-generating programme featureCounts. A quick tutorial can be found here, while the downloads and installation instructions are on the github page. Make sure you download and install the SourceForge Subread and not the Rsubread R package.

Bedtools

Bedtools is an (accurately) self-described “swiss-army knife” set of tools to analyze and select alignments using annotation or analyze and convert annotation files themselves. It is used in a conversion step in the pipeline. Downloads and installation can be found here, and a quick start can be found here.

During/after manual installation

If you are unable to use conda and need to install the software yourself, follow the instructions provided by the tool’s authors carefully. However, sometimes, tools can provide installation instructions that require the `sudo` command to install machine-wide. When working on a (cluster) computer where you have no admin rights, that is not possible. When this is the case, simply follow the instructions up until that point. When everything is unpacked, most tools (apart from Picard and Trimmomatic) will have one or more executable files. These files can be executed when you provide the path to them from the machine. For example:

```
software/samtools-1.17/bin/samtools --help
```

will execute the executable file “samtools” at location “software/samtools-1.17/bin” with the flag “--help”. This is a bit problematic, though, as the location “software/samtools-1.17/bin” is implied to be at the location from which I am working. This is called a “soft” path. As long as I stay there, there are no issues, but changing my location will mean that I need to re-direct the computer to the correct location. A solution to this is to use a longer, “hard” path, which gives the computer directions from the root (/) rather than from the working directory:

```
/path/from/root/to/software/samtools-1.17/bin/samtools --help
```

Typing such a long message each time becomes a bit of a chore after a while, though. Luckily, we don't have to. We can simply add it to our `$PATH` variable. This variable is built in and contains software that is accessed often. You can see it as a sort of taskbar, but without the fancy graphics attached to it. Doing this is pretty easy:

```
export PATH=$PATH:/path/from/root/to/software/samtools-1.17/bin/
```

This adds the location `/path/from/root/to/software/samtools-1.17/bin/` to your "taskbar" so that the programmes at that location can be easily accessed. Now, when this command is executed, you can simply call to the programme as follows:

```
samtools --help
```

Now, the one drawback of this is that you would need to do this command on every session that you start. However, this too can be easily solved. In your home directory (simply do `cd ~` to get there quickly) there is a file called `.bashrc`. This file is executed every time you open a session. Therefore, you can simply put this exporting command in this file. You can also add on more software:

```
export PATH=$PATH:\
  /path/from/root/to/software/samtools-1.17/bin/:\
  /path/from/root/to/software/FastQC/:\
  /path/from/root/to/software/bedtools2.25.0/bin/:\
  /path/from/root/to/software/<etcetera>
```

Save the file and close it. When you open a new session, this command will be executed automatically and your software will be ready to go.

TIP: Executable files are generally found in subdirectories called `bin`, and they are coloured light green if you use MobaXterm as your terminal programme.

1.2. Installing the UTP scripts

After the necessary software has been installed, we will be installing the UTP scripts. To do so, we will first need to decide where on the computer these go. When you have found or made a suitable location, you can download the scripts from github directly onto your Linux machine:

```
git clone https://github.com/AlBaarS/UTP .
```

This will download the repository onto your current location. You can specify a different location by exchanging the dot for a path to your desired directory.

Next, we need to make sure the required scripts are executable, meaning that the system understands that these scripts can perform tasks. To do so, we use the `chmod` command, which sets permissions for files and directories:

```
chmod +x UTP/bin/*
```

The +x adds permission to execute a given script. The UTP/bin/ tells the computer where to look, and the asterisk is used as a wildcard. This wildcard stands for 'zero or more characters of any kind,' which, in this context, means anything that is within the UTP/bin/ directory. We can do the same thing for the miscellaneous scripts that accompany the pipeline:

```
chmod +x UTP/misc_scripts/*
```

More about Linux wildcards can be found here.

Finally, now that all the software has been installed, you can add the installed software to the config file (described in the next chapter) and add the UTP scripts to your `$PATH` variable in your Linux system, as described in the previous subchapter:

```
export PATH=$PATH:\
  /path/to/UTP/bin/:\
  /path/to/UTP/misc_scripts/:\
  /path/to/etcetera
```

To test whether everything went successful, simply call FqC.py, utp.py, or difExp.R in the terminal with the `--help` flag, for example:

```
utp.py --help
```

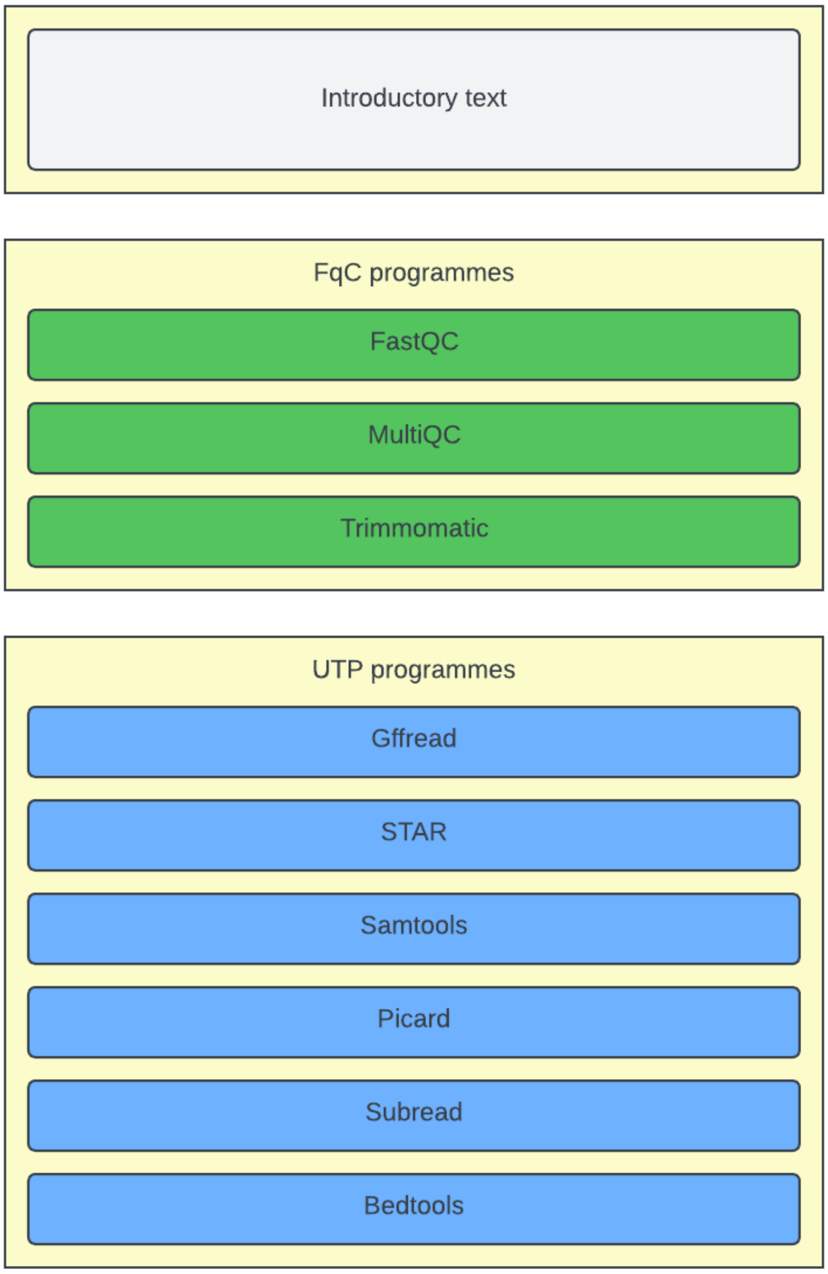
You should get a help message explaining all the possible flags for the script. These are explained in more detail in chapter 4.

2. The config files

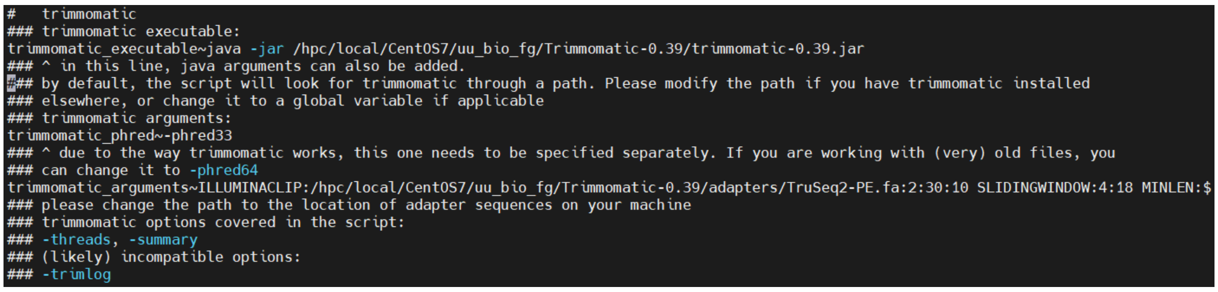
The config file (short for 'configuration file') can be used to customize your runs. Whether you want to specify different parameters in the software used or want to use different versions of the same software, the config files offer the flexibility to do so. There are some limitations, though. Here, we explain how to read the config files, how to modify them, and our recommendations/warnings.

2.1. How to read and change the UTP config file

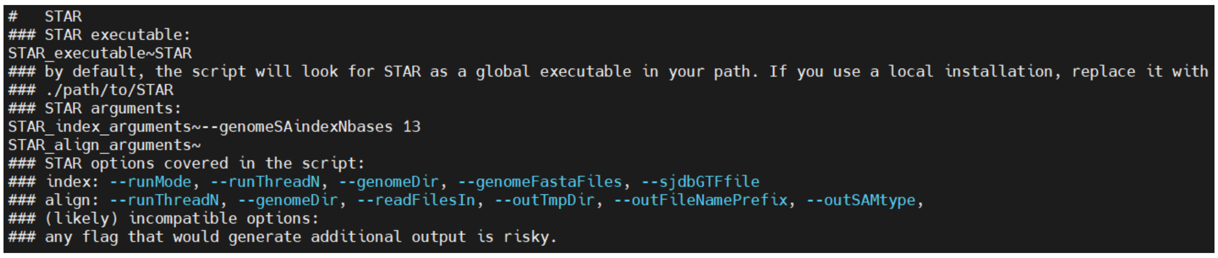
There is a single standard config file for both the FqC and UTP scripts (utp.config in the UTP/config directory). This file is divided into several sections. At the top, there is a short introduction where some of the information laid out here is repeated. Then, each programme has its own part with its options. The programmes are divided between the FqC script and the UTP script. The layout can be found in the figure below.



Each programme’s section looks somewhat similar. Let’s take a look at two example entries for Trimmomatic and the STAR aligner in the figures below:



```
#   trimmomatic
### trimmomatic executable:
trimmomatic_executable~java -jar /hpc/local/CentOS7/uu_bio_fg/Trimmomatic-0.39/trimmomatic-0.39.jar
### ^ in this line, java arguments can also be added.
### by default, the script will look for trimmomatic through a path. Please modify the path if you have trimmomatic installed
### elsewhere, or change it to a global variable if applicable
### trimmomatic arguments:
trimmomatic_phred~-phred33
### ^ due to the way trimmomatic works, this one needs to be specified separately. If you are working with (very) old files, you
### can change it to -phred64
trimmomatic_arguments~ILLUMINACLIP:/hpc/local/CentOS7/uu_bio_fg/Trimmomatic-0.39/adapters/TruSeq2-PE.fa:2:30:10 SLIDINGWINDOW:4:18 MINLEN:$
### please change the path to the location of adapter sequences on your machine
### trimmomatic options covered in the script:
### -threads, -summary
### (likely) incompatible options:
### -trimlog
```



```
#   STAR
### STAR executable:
STAR_executable~STAR
### by default, the script will look for STAR as a global executable in your path. If you use a local installation, replace it with
### ./path/to/STAR
### STAR arguments:
STAR_index_arguments~--genomeSAindexNbases 13
STAR_align_arguments~
### STAR options covered in the script:
### index: --runMode, --runThreadN, --genomeDir, --genomeFastaFiles, --sjdbGTFfile
### align: --runThreadN, --genomeDir, --readFilesIn, --outTmpDir, --outFileNamePrefix, --outSAMtype,
### (likely) incompatible options:
### any flag that would generate additional output is risky.
```

When the scripts read the config file, they take all lines that do not start with a ‘#’ and split these lines on the ‘~’ character to divide it into the **key** (before the ~) and the **value** (after the ~). The script calls upon the key and parses its value as part of the command. The keys are typically named `programme_function`, where ‘programme’ is the name of the programme and ‘function’ is the function of this option, which is either ‘executable’ or ‘arguments’ in most cases.

- **NOTE:** Do not remove or change they **keys** from the config file. **Values** can be empty in the case of argument lines, but not with executable or other config lines.

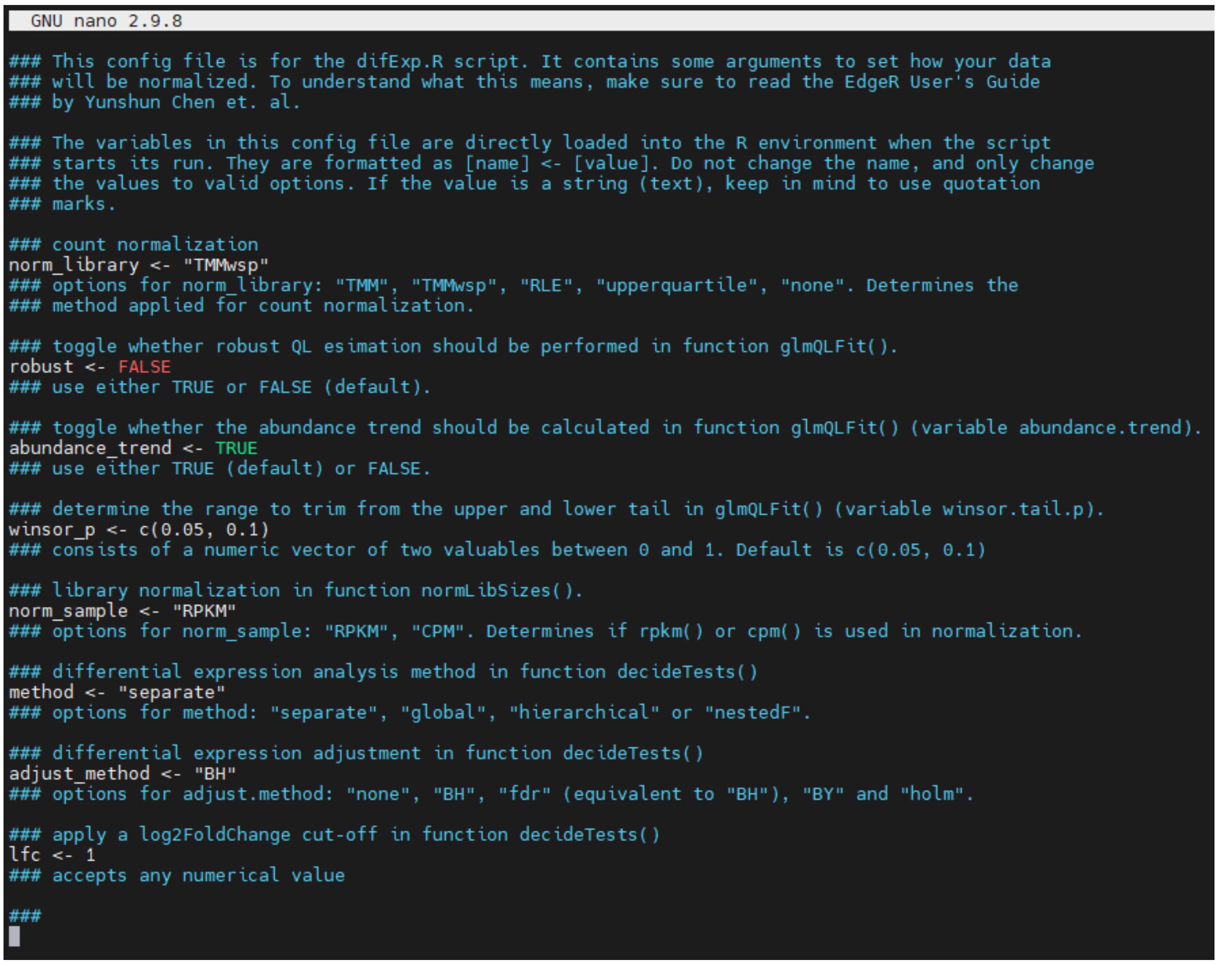
Every programme has one configurable `programme_executable`. In Trimmomatic’s case, it assumes you call upon the .jar file via `java -jar /path/to/jarfile`, while in STAR’s case, it assumes `STAR` is in your `$PATH`. However, the pipeline is built with variations in local and global installations in mind. You can simply change the values to accommodate this. So, for example, for a local installation of STAR the value would be `./path/to/STAR/`. Likewise, a global installation for Trimmomatic means you can change the value to the global variable (something like `trimmomatic` or `trimmomatic.jar`, most likely).

Next, every programme has one or more `programme_arguments`. If there are more, they are typically named as `programme_subcommand_arguments`. These values can be empty if no extra arguments are needed in your run. If you want to add additional options of that particular programme, you put them here as if you were to write them on the command line (`--flag1 option1 –flag2 option2`). The options that are included in the pipeline are listed below. Depending on the programme, there may be options that will break the pipeline, which we discourage from using when running this pipeline. These are listed at the bottom of the programme entry.

2.2. How to read and change the difExp config file

You can tweak some of the parameters used by difExp.R in the difExp.config file, located in the UTP/config/ directory. Like with the other config file, it is recommended that you copy this file elsewhere and change that rather than changing the original file. In addition, all lines starting with a ‘#’ are also comment lines, meaning that they are not interpreted by the programme, but it is still recommended to not change these.

The config file itself is rather small, and contains only eight options to change, as can be seen below. Directions to software or packages is not necessary; difExp.R only uses R packages, which are automatically installed when using the script for the first time. These options are simply taken as variables when running difExp.R, and are therefore defined as such in the config file. To change them, you can change the value behind the arrow (<-) to your preferred value. Note the use of quotation marks where necessary. For a comprehensive explanation as to what these variables or functions do, we recommend checking out the edgeR handbook and/or the R documentation for each of these functions.



```
  GNU nano 2.9.8

### This config file is for the difExp.R script. It contains some arguments to set how your data
### will be normalized. To understand what this means, make sure to read the EdgeR User's Guide
### by Yunshun Chen et. al.

### The variables in this config file are directly loaded into the R environment when the script
### starts its run. They are formatted as [name] <- [value]. Do not change the name, and only change
### the values to valid options. If the value is a string (text), keep in mind to use quotation
### marks.

### count normalization
norm_library <- "TMMwsp"
### options for norm_library: "TMM", "TMMwsp", "RLE", "upperquartile", "none". Determines the
### method applied for count normalization.

### toggle whether robust QL esimation should be performed in function glmQLFit().
robust <- FALSE
### use either TRUE or FALSE (default).

### toggle whether the abundance trend should be calculated in function glmQLFit() (variable abundance.trend).
abundance_trend <- TRUE
### use either TRUE (default) or FALSE.

### determine the range to trim from the upper and lower tail in glmQLFit() (variable winsor.tail.p).
winsor_p <- c(0.05, 0.1)
### consists of a numeric vector of two valuables between 0 and 1. Default is c(0.05, 0.1)

### library normalization in function normLibSizes().
norm_sample <- "RPKM"
### options for norm_sample: "RPKM", "CPM". Determines if rpkm() or cpm() is used in normalization.

### differential expression analysis method in function decideTests()
method <- "separate"
### options for method: "separate", "global", "hierarchical" or "nestedF".

### differential expression adjustment in function decideTests()
adjust_method <- "BH"
### options for adjust.method: "none", "BH", "fdr" (equivalent to "BH"), "BY" and "holm".

### apply a log2FoldChange cut-off in function decideTests()
lfc <- 1
### accepts any numerical value

###
```

2.3. Do's & Don'ts

- **DO** make a copy of the default config if you want to change the parameters in a run rather than changing the default config file.
- **DO** double-check the options you can and need to put into a programme. Some options in the config file are mandatory for it to work.
- **DO** check the paths of executable and other necessary files in the config file when running the first time or when changing them.
- **DO** use hard paths when you need to specify paths in your config file.
- **DON'T** delete lines from the config file. While the comment lines are not necessary for it to function, they help you in properly changing the configuration and finding the right parts to modify.

3. Pipeline assumptions

In order to effectively use this pipeline, there are some things you need to know. Please check these things before running the pipeline and make sure your files fulfill all the listed criteria. After all, it is better to prevent the pipeline from breaking after several hours of running.

3.1. The genome and annotation file must have the same name.

In order to properly match the genome and annotation file, they must have the same name. The UTP script splits names on a dot and tries to match the first part (i.e. before the first dot). For example:

- Homo_sapiens_assembly_v38.fna
- Homo_sapiens_assembly_v38.gff3
- S_cerevisiae.GC020292.fasta.gz
- S_cerevisiae.annot_2018.gtf
- A_thaliana.fasta
- A_thaliana.genes.bed

These genome files will all match their respective annotation file. Keep this in mind as you name your files, because using files like this may cause trouble:

- B.bassiana.fasta
- B.bassiana.annot.gff

In this example, the pipeline will use 'B' as the name, and if you have other files named this way that share a letter, things will break.

Files like this will not work:

- Sus_scrofa_genome.v4.fasta
- Sus_scrofa.annotation.v4.gff

Because the strings before the first dot do not match (Sus_scrofa_genome v/s Sus_scrofa). Also keep in mind to make sure different genome files and annotation files won't match each-other. The names are case sensitive.

3.2. There is one annotation file per genome file.

If you have more than one annotation file per genome file (e.g. when you want to compare multiple types of regions), it is recommended to merge them into a single file and sort them. If you have more than one genome file per annotation file, it is recommended to copy your annotation file and rename the copy to match the other genome, or perform separate runs for both.

3.3. The chromosome/scaffold names first element must match.

The chromosome/scaffold names in the genome file can consist of multiple words, as long as the first word (separated by a single whitespace) matches the chromosome/scaffold names in the annotation file. So, for example:

- Chromosome name (genome file): >scaffold01 unplaced Saccharomyces cerevisiae
- Chromosome name (annotation file): scaffold01

These entries will match. However, entries like this:

- Chromosome name (genome file): >Chr02_Homo_sapiens version 38
- Chromosome name (annotation file): Chr02

These entries will not match, and the pipeline will stop.

3.4. Fasta headers and annotation do not contain ‘@’ in the chromosome name.

Within the first part of the header, as specified in the previous section, cannot contain an ‘@.’ This is the symbol that is used when merging genomes and annotation to link the name of the genome to the respective chromosome like this:

Chromosome@Genome_Name

When separating the names later, the script uses the ‘@’ as a cutting point. If there is one or more ‘@’s’ in the name, the script will cut incorrectly, causing things to break.

3.5. Paired-end reads have R1/R2 in their name.

The read files have their full name and (if paired end) R1/R2 in their name/ID before the first occurence of a dot. This assumption is necessary to properly pair the right files with each-other and to name the output correctly. If this is not the case, we recommend changing any dots separating the name and R1/R2 to an underscore.

3.6. All read files are compressed with the same method.

If the read files are compressed, they are all compressed with the same method (either gzip or bzip2). If you are unsure of the method used, gzipped files have the .gz extension, while bzip2 files have the .bz2 extension.

3.7. Unpaired reads are marked in the filename.

For the main UTP script: if you want to run the pipeline paired-end and have a mix of paired/unpaired read files in your input, make sure that the unpaired files have the 'unpaired' string in their file or directory name. The script will filter those out from analysis. The easiest way to get this is to use the Fastq Cleaning script which is provided with the pipeline, if you want to trim your data.

3.8. Gene counts are formatted as one count per measured feature.

For the difExp.R script, it is assumed that the counts are in a format of one count per feature. If you used the UTP script, this is done automatically, but it should be taken into account if you get your counts from a different source.

By default, the configuration of the UTP script is that featureCounts will measure at the transcript level (i.e. take the whole gene), but you can change this to exon, CDS, etc. As long as these features are in your annotation file, of course.

4. Pipeline usage

With your pipeline fully installed, your files checked, and your config file configured, you are ready to run the actual pipeline. It consists of three scripts: The Fastq-cleaner script (FqC), the Unnamed Transcriptomics Pipeline (UTP), and the Differential Expression analysis script (difExp). Here, we explain how to use these scripts and how to use the given options.

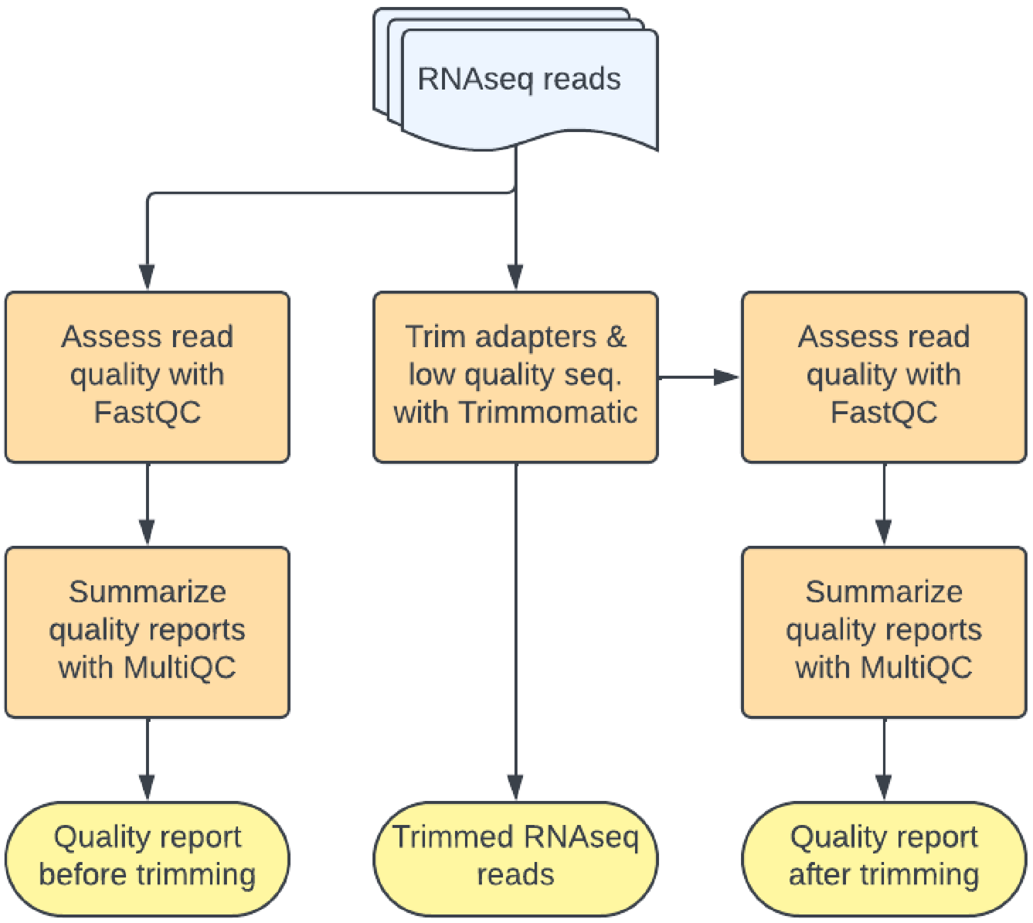
4.1. The FqC script

The FqC script is a short pipeline to clean and organize your data for the main pipeline. It is not mandatory if your data is already cleaned and ready to go, but generally recommended. It incorporates the programmes FastQC, MultiQC, and Trimmomatic to perform quality control and trim adapter sequences from the read files, and works with both paired-end and not paired-end data. It has the following options:

- `--reads [directory] OR [read file] OR [read file 1] [read file N]`
  With this option, you can specify the path to your read file(s). It accepts either a single path to a directory that contains the readfiles, multiple paths to files (separated by a single whitespace), or a single file. Note that it will treat the single file as a single sample, and does not support different samples being pooled into a single file.
- `--outdir [directory]`
  Here, you can specify the directory where you want the output to go. In this case, the output will be trimmed fastq files.
- `--tmpdir [directory]`
  Here, you can specify the directory to store temporary files in that the pipeline makes. Within this directory, the pipeline will make a directory of its own called `FqC_tmpdir_<timestamp>_tmp/`, with 'timestamp' being the time that the pipeline starts (roughly). By default, this directory and its contents will be deleted at the end of the run, but you can save the contents if you wish.
- `--keep_tmp_data`
  If this flag is specified, the temporary directory will be saved at the specified temporary directory. Useful if you want to debug your run.
- `--copy_tmp_data [directory]`
  If this flag is specified, the temporary directory will be moved to a specified directory. Useful if you use this on a cluster with a dedicated `$TMPDIR` and want to debug your run.
- `--config [path/to/config_file]`
  Specify a config file. Not necessary if you want to use the default config file. See Chapter 2 for more information on the config files.
- `--threads [integer]`
  The number of threads allocated to the programmes that support multithreading. By default, this is 2.
- `--single_end`
  Specify that your data is single-end sequenced rather than paired-end, which is the default.
- `--verbose`: If specified, the pipeline will print more info on what it is doing, accompanied by timestamps. A useful feature if you want to see how long running each process takes on your file.
- `--help`
  Prints out all of the options with a short summary of what they do and how to use them and then exits.

- `--version`
  Prints out the programme version and then exits.

The script performs quality control before and after the trimming step, meaning that you can easily compare between before and after and adjust your workflow accordingly. Keep in mind that, while multiple FastQC runs are performed, the arguments specified in the config file will be used both pre and post-trimming, as do the arguments for MultiQC. In fact, MultiQC is also used in the UTP script. Keep this in mind as you select your parameters. Depending on the quality of your data, multiple runs may be needed to get the parameters right before moving on to the alignment stage.


RNAseq reads
Assess read quality with FastQC
Trim adapters & low quality seq. with Trimmomatic
Assess read quality with FastQC
Summarize quality reports with MultiQC
Summarize quality reports with MultiQC
Quality report before trimming
Trimmed RNAseq reads
Quality report after trimming

A typical command to run FqC would be:

```
FqC.py \
  --reads data/testing_data/RNAseq/Ophiocordyceps/head/ \
  --outdir data/trimmed_reads/Ophiocordyceps/LM/head \
  --tmpdir $TMPDIR \
  --threads 8 \
  --verbose \
  --config home/utp_hpc.config \
  --copy_tmp_data data/tmp/
```

This particular command was used on a cluster, we made a copy of the config file which contained the necessary settings and made sure our temporary data was moved to a location for storage, as it would be deleted in the system's `$TMPDIR` space.

- **TIP**: If you are running this software on a cluster too, put the commands in a bash script and let it run from the `$TMPDIR`. Sometimes, temporary files are made by programmes that are stored in the directory the script is run from – which is likely your home directory by default.

4.2. The UTP script

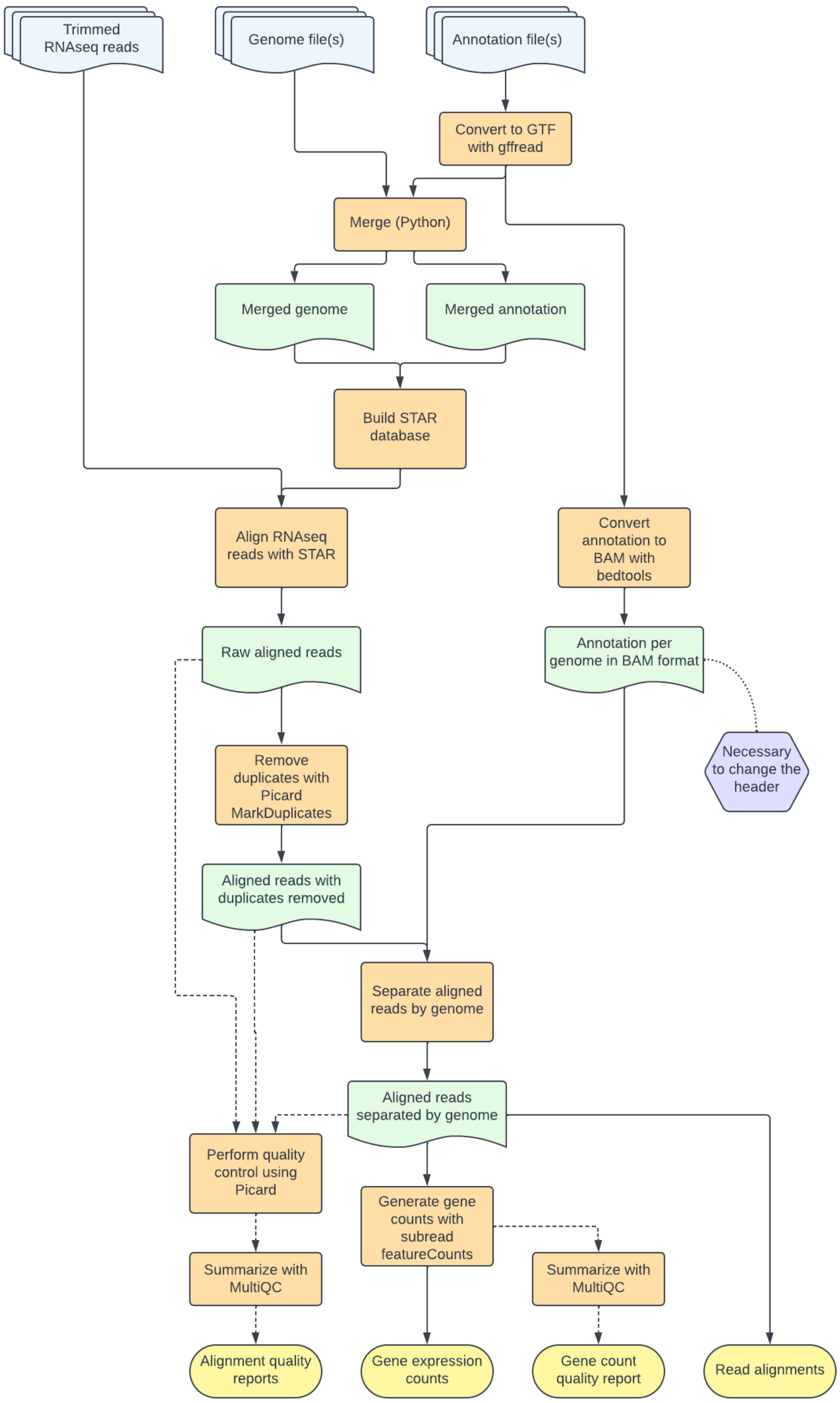
The alignment and data formatting that you need to perform differential expression is done with the UTP script. It merges the genomes and annotation if multiple genomes are specified and aligns RNAseq reads using STAR to that merged genome. Then, it removes duplicate mappings using Picard MarkDuplicates, separates the alignments with samtools, and creates gene counts from the aligned data using Subread's featureCounts tool. It uses Gffread and Bedtools for some conversion steps in-between. Finally, it uses Picard and MultiQC for quality control between the different steps. It has the following options:

- `--reads [directory] OR [read file] OR [read file 1] [read file N]`
  With this option, you can specify the path to your read file(s). It accepts either a single path to a directory that contains the readfiles, multiple paths to files (separated by a single whitespace), or a single file. Note that it will treat the single file as a single sample, and does not support different samples being pooled into a single file.
- `--genome [directory] OR [genome file] OR [genome file 1] [genome file N]`
  With this option, you can specify the path to your genome fasta file(s). It accepts either a single path to a directory that contains the genome files, multiple paths to files (separated by a single whitespace), or a single file. If multiple genome files are specified, they will be merged and a merged alignment will be performed. The number of genome and annotation files submitted must be the same, and they must have a matching name (see Pipeline Assumptions).
- `--annotation [directory] OR [annotation file] OR [annotation file 1] [annotation file N]`
  With this option, you can specify the path to your annotation file(s). It accepts either a single path to a directory that contains the annotation files, multiple paths to files (separated by a single whitespace), or a single file. If multiple annotation files are specified, they will be merged and a merged alignment will be performed. The number of genome and annotation files submitted must be the same, and they must have a matching name (see Pipeline Assumptions).
- `--outdir_bam [directory]`
  Here, you can specify the directory where you want the output to go. In this case, the output will be binary alignment map (BAM)-files.
- `--outdir_counts [directory]`
  Here, you can specify the directory where you want the output to go. In this case, the output will be tab-delimited text files containing the gene count information. Keep in mind that the gene counts will be put in a single file per run and per genome, not per sample.
- `--tmpdir [directory]`
  Here, you can specify the directory to store temporary files in that the pipeline makes. Within this directory, the pipeline will make a directory of its own called `UTP_tmpdir_<timestamp>_tmp/`, with 'timestamp' being the time that the pipeline starts (roughly). By default, this directory and its contents will be deleted at the end of the run, but you can save the contents if you wish.

- `--keep_tmp_data`
  If this flag is specified, the temporary directory will be saved at the specified temporary directory. Useful if you want to debug your run.
- `--copy_tmp_data [directory]`
  If this flag is specified, the temporary directory will be moved to a specified directory. Useful if you use this on a cluster with a dedicated `$TMPDIR` and want to debug your run.
- `--config [path/to/config_file]`
  Specify a config file. Not necessary if you want to use the default config file. See Chapter 2 for more information on the config files.
- `--old_picard`
  If specified, it will use the old Picard syntax. You can use this tag if you want to use an older installation of Picard in your runs.
- `--threads [integer]`
  The number of threads allocated to the programmes that support multithreading. By default, this is 2.
- `--single_end`
  Specify that your data is single-end sequenced rather than paired-end, which is the default.
- `--verbose`
  If specified, the pipeline will print more info on what it is doing, accompanied by timestamps. A useful feature if you want to see how long running each process takes on your file.
- `--help`
  Prints out all of the options with a short summary of what they do and how to use them and then exits.
- `--version`
  Prints out the programme version and then exits.

The genome merging functionality is a core mechanism in this pipeline, which enables users to look at RNAseq data with mixed reads from different organisms. It is not built for metagenomic studies, but rather for studies in which there are mixed reads with a small number of known organisms, such as parasite-host and symbiotic experiments. The merging itself is simply done in Python, where the fasta files are simply concatenated together with each scaffold being tagged for its respective genome (where >scaffold1 becomes >scaffold1@genome.fasta). The annotation is converted to GTF format using Gffread with the -F flag (baked in) so that no attributes are lost in the conversion progress and then merged as well.

There are several steps for quality control present to assess the quality of the alignments. First on the raw alignment output, then on the bam file with duplicates removed, and finally on the split bamfiles. If you are running the pipeline on a single genome (meaning there is no merging), it is only performed on the alignment files before and after the removal of duplicate alignments.


Trimmed RNAseq reads
Genome file(s)
Annotation file(s)
Convert to GTF with gffread
Merge (Python)
Merged genome
Merged annotation
Build STAR database
Align RNAseq reads with STAR
Convert annotation to BAM with bedtools
Raw aligned reads
Annotation per genome in BAM format
Necessary to change the header
Remove duplicates with Picard MarkDuplicates
Aligned reads with duplicates removed
Separate aligned reads by genome
Aligned reads separated by genome
Perform quality control using Picard
Generate gene counts with subread featureCounts
Summarize with MultiQC
Summarize with MultiQC
Alignment quality reports
Gene expression counts
Gene count quality report
Read alignments

A typical command to run the UTP would be as follows:

```
utp.py \
  --reads data/testing_data/trimmed_reads/*.fastq \
  --genome data/refseq/Ophiocordyceps/ data/refseq/Camponotus \
  --annotation data/refseq/Ophiocordyceps/ data/refseq/Camponotus \
  --outdir_bam data/alignment/Ophiocordyceps/LM/head/ \
  --outdir_counts data/counts/Ophiocordyceps/LM/head/ \
  --tmpdir data/tmp/ \
  --keep_tmp_data \
  --config home/utp_hpc.config \
  --verbose \
  --threads 8
```

This particular command was used on a cluster, we made a copy of the config file which contained the necessary settings and made sure our temporary data was moved to a location for storage, as it would be deleted in the system's `$TMPDIR` space. Note the use of a wildcard '*' for the input. The wildcard in question stands for "zero or more characters of any kind," and is followed by '.fastq.' This means that we select all files or directories at the given location that end with .fastq. By default, the script will look for this automatically if you simply give it a directory to take the input from.

- **TIP**: If you are running this software on a cluster too, put the commands in a bash script and let it run from the `$TMPDIR`. Sometimes, temporary files are made by programmes that are stored in the directory the script is run from – which is likely your home directory by default.
- **TIP**: Make use of Linux wildcards when necessary: https://tldp.org/LDP/GNU-Linux-Tools-Summary/html/x11655.htm

4.3. The difExp script

The difExp script calculates differential expression based on the counts generated by the UTP script. It loads the counts data and the information you provide on the command line to organize the right samples with the right condition, normalizes, and them compares them using the EdgeR package. After that, it saves the results in tables and figures revealing which genes are differentially expressed between the conditions set. The script has the following options:

- `--input_counts [list of files.counts]`
  You specify your input files with this flag. There is no hard limit on the number of files to compare, but it requires a minimum of 2. Please only submit the count files for the conditions you wish to compare directly (e.g. all files belonging to a time range). If there are different tests to be compared (e.g. expression over a timerange, expression in organ1 vs organ2, expression with or without infection), perform a run for each test individually. Make sure there are **no whitespaces** in between the commas, otherwise, the script will not interpret your list correctly. For example: `Ecoli_infection.counts,Saureus_infection.counts`. The script will pool all the entries together into a single file, with the columns appearing in the order as they do in the files and in the order the files are submitted.
- `--output_dir [path/to/directory]`
  Here you specify the path to your output directory. Since multiple files will be generated, a single output file cannot be specified. If you want to prevent files from being overwritten, make sure the directory is empty. We recommend to make a dedicated output directory for each test you wish to perform.

- `--name [name]`
  This flag enables you to specify a prefix for your filenames. By default, your files will be called 'DE_analysis_<rest_of_filename.suffix>'. Using this flag, you can change that to a more informative filename '<name>_<rest_of_filename.suffix>'.
- `--replicates [list of no. replicates]`
  With this flag, you specify how many replicates each condition has. This must be done for every condition individually. For example, if you have three conditions to compare with five samples each, you specify `--replicates 5,5,5`. If you have an unequal number of replicates per condition, that is not an issue. The number of entries in this list (3 in the example) has to be equal to the number of entries specified in `--conditions`.
- `--conditions [list of conditions]`
  With this flag, you specify the names of your conditions that you compare. We recommend keeping these short (<10 characters). For example, if you compare expression between organs, you can submit `--conditions spleen,liver,kidney`. The number of entries in this list (3 in the example) has to be equal to the number of entries specified in `--replicates`.
- `--samples [list of sample names]`
  This flag comes in handy if you wish to change the sample names from those found in your counts files. If not specified, the script will simply take the names provided in the counts file and remove the path (if any).
- `--omit_genes [file with geneIDs]`
  With this flag, you can specify genes to be omitted from the analysis. It expects a file with one geneID per line and no other information or headers. Make sure the geneIDs found here match the ones in the counts files exactly.
- `--config [path/to/file.config]`
  Specify a config file. Not necessary if you want to use the default config file. See Chapter 2 for more information on the config files.
- `--verbose`
  Specify if you wish to run the script in verbose mode. This prints more information about what's going on behind the scenes.
- `--save_ggplot_obj`
  Specify whether you want to save the raw ggplot2 objects to .rds files. These files can be imported into R with the readRDS() function. If you want to summarize some of the plots generated by the pipeline into a single figure, you can use this function to get the plot's raw R data and then merge them into single figures using the ggpubr package. Keep in mind that you need the tidyverse packages installed for all plots, and the ggpointdensity package for the scatterplots specifically in order for the plots to work correctly.
- `--debug`
  Specify if you wish to run the script in debug mode. This prints the lines of code and highlights where things are going wrong (at least, to the best of R's ability). Not necessary while performing your tests, but a useful tool if something keeps going wrong.
- `--help`
  Prints out all of the options with a short summary of what they do and how to use them and then exits.

There is one fundamental thing that must be explained before diving into the output. The way the script calculates differential expression can be explained as follows: Calculate -A + B. Is the result close to 0, then the expression is similar. If the result is negative (i.e. A is bigger), then the gene is downregulated in B compared to A. If the result is positive (i.e. B is bigger), then the gene is upregulated in B compared to A. The same goes for other comparisons if more than two conditions are specified (e.g. -A + C, -B + C). Of course, this explanation is very simplified, but it is important to know when interpreting your results.

A typical command to run difExp.R would be as follows:

```
difExp.R \
```

- `--input_counts data/counts_curated/Ophiocordyceps/head/Ophcf2.LT25.counts,data/counts_curated/Ophiocordyceps/head/Ophcf2.LM.counts,data/counts_curated/Ophiocordyceps/head/Ophcf2.DM.counts,data/counts_curated/Ophiocordyceps/head/Ophcf2.24D.counts \`
- `--output_dir data/dif_exp_analysis/timepoints/Ophiocordyceps/ \`
- `--name Ophcf2_head \`
- `--replicates 4,5,5,5 \`
- `--conditions LT25,LM,DM,24D \`
- `--samples OLT25H1,OLT25H2,OLT25H3,OLT25H4,OLMH1,OLMH2,OLMH3,OLMH4,OLMH5,ODMH1,ODMH2,ODMH3,ODMH4,ODMH5,O24DH1,O24DH2,O24DH3,O24DH4,O24DH5 \`
- `--test_method QLFTest \`
- `--save_ggplot_obj \`
- `--omit_genes data/expression_levels/Ophcf2_genes_rm.txt \`
- `--verbose`

Here, we specified our input counts files one by one. Note that there are no whitespaces between the commas and the next entry. This is due to the differences between R and Python in input flexibility; Python can understand multiple entries (separated by a single whitespace) can belong to a single argument, while R cannot. Therefore, in this case, a wildcard would also not work. Next, we specified the output directory and output prefix, the latter of which is not mandatory, but recommended to prevent accidentally overwriting different results. The conditions and replicates specify which columns in the data files belong to which condition and to which replicate. In this case, we had four input files, where the first file had four columns (as one sample was discarded) and the others had five. The input files are merged into a single table, and these parameters make sure the right data is associated with the right condition. Next, the sample names are specified. By default, the script will use the file and column names to create its own sample names, but these are quite long. It is therefore recommended that you specify them manually. After that, the testing method is specified. This parameter refers to which edgeR function is used for testing. By default, this is the QLFTest() function, but this can be changed to the QLFTreat() or glmLRT() functions. These all use different mathematics to determine differential expression. For a proper explanation, it is recommended to check out the edgeR manual. To give a quick comparison between the three, they mostly differ in terms of strictness, with QLFTreat() being the most strict and glmLRT() being the least strict. Which one to use depends on the research question and the quality of your data. Alternatively, you may choose to tweak some of the parameters in the config file instead (such as decreasing the P-value or different normalization). Finally, we save the generated plots to separate files so they can be put into figures for publication, removed some of the genes with low expression, and ran with `--verbose` to get a better overview of the results as they are processed.

4.4. Miscellaneous scripts

Finally, the pipeline also comes with six miscellaneous scripts to solve some of the smaller issues that I encountered when running the pipeline. First, a script called change_gene_length.py. This script was made to change the 'length' column in a featureCounts counts file (generated by utp.py) from the full gene length to the mRNA length, consisting solely of exons. If normalizing by mRNA length rather than gene length is desirable due to different transcripts, you can change it using this script, needing only the input counts file and the annotation file in GFF v3 format. You also need to specify an output directory to put the new file into. Finally, you may need to specify how many lines belong to the input file's header – lines which contain column and input information, usually preceded by "#". In the case of an unmodified counts file from featureCounts/utp.py, this is two lines, which is set to two by default.

Second, there is cts.py. CTS stands for Calculate Transcriptome Size, which is what the script does. Simply submit an annotation file and specify for what kind of element (gene, exon, intron, CDS, etc.) you want to calculate the total size of. It simply prints the number of bases covered by the specified element.

Third, there is ela.R, which stands for Expression Level Analyzer. This script uses your count files to select genes with low expression to be eliminated from analysis. The required count files, output directory, conditions, replicates, and sample names (if desired) are submitted in the same way as for difExp.R. For normalization, the script can use either RPKM (default) or CPM. The expression is averaged per gene between all columns belonging to a submitted condition. To investigate your data, you run with `--mode testing`, which is the default. Running in this mode, the script returns plots detailing how many genes are left at a given cut-off value, which will allow you to gauge what kind of cut-off you want to use. In addition, the exact values projected in this plot can be found in the accompanying table. When you have settled on a cut-off, you can run the script with `--mode filter`, which will return a txt file with a list of geneIDs that do not meet the threshold in any of the averaged conditions. Note that this means that some genes with expression that meets the threshold in only one or two replicates may therefore also be excluded.

Fourth, the packages.R script simply makes sure that all the necessary packages are installed. It has no arguments to be specified. This way, if you are just setting up for the first time, you can install the packages easily without having to run difExp.R. Keep in mind that it does not update the packages if they are installed already. To do so, you can manually go into the R version that you use (simply by typing 'R' in the terminal) and manually install/update the necessary packages. If you use install.packages(<package>), this will update any installed package under that name if it is installed already.

Fifth, the remove_counts.py script enables you to easily remove columns from the counts file if necessary. You specify the input and output files and which samples need to be removed. The sample specification is done numerically. You can remove multiple samples at once.

Finally, there is the rename_geneIDs.py script. This, as the name suggests, re-names the used geneIDs. In my case, I generated counts at a transcript level, which therefore adopted the transcript ID as the geneID. This caused issues when I was comparing the data to the annotation, as these IDs did not match. This script changes transcript IDs to geneIDs, and can also be used on exonIDs, CDS-IDs, etc. For the input, the file of interest and annotation have to be provided. The input file can be a counts file, but also a different kind of table, or the output from ela.R. With this versatility in mind, the separator (i.e. what separates the different columns) must also be specified. By default, this is a comma. Counts files are tab-delimited, which can be set with `--separator "\t"`. A list of IDs as made by ela.R has no separators, as it is a single column. These files can be processed using `--separator "none"`. Also keep in mind how many header lines files have; check this by opening the file in a text editor or by printing the top lines on Linux using the `head` command.

5. Pipeline errors

There are many things that can go wrong when running a programme or script. File formats may be slightly off, data may be missing, etc. To best prevent mistakes from occurring, or to solve them in the most efficient way, it is wise to read a programme's documentation. We assume that you have read this manual faithfully so far and have not treated it like a 'terms and conditions' document when signing up to a website. In either case, we will list out the errors and warnings that the scripts can give, what they mean, and how to fix them. For some of the possible errors caused by the software used in the pipeline, we have also provided an explanation and how to fix them in the next chapter.

5.1. FqC/UTP errors and warnings

Errors and warnings described here can occur in both scripts.

- `UTP/FqC Error: Files not found; aborting.`
  The script cannot find any read, genome, or annotation files at the specified location. Make sure your path is correct and exists, and that all files end in a suffix typically associated with their type.
- `UTP/FqC Error: Cannot find config file: <path>` and `UTP/FqC Error: Cannot find config file at <config_file>. Exiting...`
  A config file cannot be found at the specified location. Make sure that your path is correct and exists. If you have not specified the `--config` flag, that means that the script cannot find the default config file. If you have moved or copied either the script or the config file, this can happen. The script assumes that it is located in a folder called '`bin`' and that the config is in the directory `../config/` relative to the script location. Make sure that this is the case, or manually specify the location of your config file.
- `UTP/FqC Error: Cannot find output directory.`
  The script cannot find the specified output directory. Make sure that your path is correct and exists; the script does not make this directory for you.
- `UTP/FqC Error: Cannot find temporary directory.`
  The script cannot find the specified temporary directory. Make sure that your path is correct and exists; the script does not make this directory for you.
- `UTP/FqC Error: Cannot find "R1" or "R2" in file name <file>, removing this file from analysis.`
  The script cannot find 'R1' or 'R2' in a file that was submitted as a read file, assuming a paired-end run. If you are using single-end reads, include the `--single_end` flag in your run. If your reads are paired-end, make sure that R1/R2 is in the file name before the first occurrence of a dot (see Pipeline Assumptions).
- `UTP/FqC Warning: Output directory not empty. Existing files may be overwritten.`
  The script warns that there are other files or directories present within the specified output directory. If any files present match the name of one of the output files (e.g. from earlier runs), they will be overwritten. Keep this in mind if you want to save your previous output.

5.2. FqC errors and warnings

- `FqC Error: unequal number of paired read files.`
  In a paired-end run, the number of read files with 'R1' does not match the number of read files with 'R2.' Make sure you have an even number of read files in a paired-end run, and that all of them have R1 and R2 in the name (see Pipeline Assumptions).

5.3. UTP errors and warnings

- `UTP Error: Could not find match for <file>`
  When multiple genomes are submitted, the script will merge them for alignment downstream. It will check the annotation files submitted for a matching name. This error message means that that has failed, and for which file. Make sure that the genome files and annotation files have a matching filename (see Pipeline Assumptions).
- `UTP Error: number of genome files != number of annotation files, exiting.`
  The number of genome files and annotation files submitted is not equal. Make sure you submit an equal number of genome files and annotation files (see Pipeline Assumptions).

6. Software errors

This pipeline uses various programmes to do the heavy lifting. The pipeline itself is more of a standardized way to instruct those programmes what to do. However, even then there may be issues with the submitted files that only become apparent further downstream. In this chapter, we provide tips to solve any software errors you could encounter and list a few specific examples that we ran into ourselves. Keep in mind though that we cannot possibly predict every single error message that each programme can generate.

6.1. Best Practice to solving errors

To best solve your issues, there are a few things you can do when running things. We recommend doing these for your first run(s), when it's more likely things go wrong.

1. Save your temporary data.
   a. If you need to check any intermediary files, you need to tell the pipeline to save them. See the Pipeline Usage chapter for more information.
2. Use the `--verbose` tag in your run.
   a. When running verbose, a lot of information about what is going on behind the scenes is printed, including the commands issued to the incorporated software. This makes it easier to spot a mistake, as well as benchmark the pipeline on your own data.

6.2. Specific examples

Here, we list a few examples of software errors that we ourselves encountered when running the pipeline.

- featureCounts - `ERROR: failed to find the gene identifier attribute in the 9th column of the provided GTF file.`
  - This error is the result from one or more entries missing the 'gene_id' attribute. You can bypass this error by including the flag '`-g transcript_id`' in the config file, or use another attribute if it suits your experiment better.
  - If you still find the error with other attributes, that means that some entries may be missing a lot of their information. Investigate the converted annotation file (found at `<tmpdir>/genome_index/<genome>.annotation.gtf`) for missing entries. If possible, we recommend omitting entries that lack much information. Alternatively, you can add the field manually or through code to the entries that miss it.
- Picard (MarkDuplicates) - `java.io.IOException: Disk quota exceeded.`
  - This error message is somewhat elusive. It says that it cannot write something to the disk. In reality, this error has two factors that cause it. 1. Picard has not enough RAM and is writing excess data to the disk. 2. The disk in question is not letting Picard write said data onto it. This issue is simply solved by allocating more RAM.
  - For my personal experiments (2 genomes, total ~ 300 MB filesize, read files ~1.2 GB filesize when compressed), about 24 GB allocated to Java should do the trick. However, keep in mind that, when you allocate memory to a Java programme with `-Xmx24g`, that Java takes an additional ~2-4 GB of memory to run the virtual environment that it creates. So, if you allocate 24 GB, make sure you have at least 28 GB of RAM actually available. Much more than that is usually not necessary though.

6.3. Other possible problems

There are also some issues one may run into when running the pipeline that may not be immediately apparent. Here, we list some issues that we have found during testing and how to solve/mitigate them.

- STAR alignment taking very long on small samples and using a lot of computing power
    - This may have multiple causes, but it is most likely due to a low **mapping rate**, i.e. the speed at which a proper alignment is found. This can be due to low sequencing quality, contamination, or a sample containing reads from multiple species whose genomes were not included in the alignment. STAR will try very hard to find a place to align a read, and if it fails initially, it will take more resources and time to try again. When there are many reads that do not align to any of the used genomes, this will increase time substantially. In our own case, it increased the total running time on two samples from ~1.5 hours to almost 5.5 hours.