
cellhub

Release 0.1

Sansom lab

May 11, 2021

CONTENTS

| | | |
|----------|-------------------------------------|-----------|
| 1 | Installation | 1 |
| 1.1 | Installation | 1 |
| 2 | Overview | 2 |
| 2.1 | Workflow Overview | 2 |
| 2.2 | Workflow Diagram | 4 |
| 2.3 | Usage | 6 |
| 3 | Examples | 7 |
| 3.1 | IFNb PBMC example | 7 |
| 4 | Coding guidelines | 10 |
| 4.1 | Coding Guidelines | 10 |
| 5 | Pipeline documentation | 13 |
| 5.1 | Pipeline ambient rna | 13 |
| 5.2 | Pipeline Cellranger Multi | 14 |
| 5.3 | Pipeline Cell QC | 15 |
| 5.4 | Pipeline Emptydrops | 17 |
| 5.5 | Pipeline Velocity | 18 |
| 5.6 | Pipeline celldb | 19 |
| 5.7 | Pipeline fetch cells | 20 |
| 5.8 | Pipeline Integration | 22 |
| 5.9 | Pipeline Export | 24 |
| 6 | Indices and tables | 26 |
| | Python Module Index | 27 |
| | Index | 28 |

INSTALLATION

1.1 Installation

1.1.1 Dependencies

Core dependencies include:

- Cellranger (from 10X Genomics) ≥ 6.0
- Python3
- R ≥ 4.0
- (Latex)

1.1.2 Installation

1. Install the cgat-core pipeline system following the instructions here <https://github.com/cgat-developers/cgat-core/>.
2. Clone and install the cellhub-devel repository e.g.:

```
git clone https://github.com/COMBATOxford/cellhub-devel.git
cd cellhub-devel
python setup.py develop
```

Note: Running “python setup.py develop” is necessary to allow pipelines to be launched via the “cellhub” command.

3. In the same virtual or conda environment as cgat-core install the required python packages:

```
pip install -r cellhub-devel/python/requirements.txt
```

4. To install the required R packages (the “BiocManager” and “devtools” libraries must be pre-installed):

```
Rscript cellhub-devel/R/install.packages.R
```

OVERVIEW

2.1 Workflow Overview

2.1.1 Philosophy

Cellhub is designed to efficiently parallelise the processing of large datasets. Once processed different data slices can be easily extracted directly from the original matrices, aligned and exported for downstream analysis. At the heart of operations is an sqlite database which warehouses the experiment metadata and per-cell statistics.

The workflow can be divided into seven main steps.

2.1.2 1. Quantitation of per-channel libraries

The workflow begins with *pipeline_cellranger_multi.py*. Input 10X capture channel library identifiers “library_id” and their associated fastq files are specified in the *pipeline_cellranger_multi.yml* configuration file. The reads from the different libraries will be mapped in parallel.

Note: The channel library is considered to be the fundamental “batch” unit of a 10X experiment. Cells captured from the same channel are exposed to the same ambient RNA. Separate genomic libraries are prepared for each 10x channel.

- It is recommend to inspect patterns of ambient RNA using *pipeline_ambient_rna.py*.
- Per-channel velocity matrices can be prepared using *pipeline_velocity.py*.
- Cell identification can also be performed with *pipeline_emptydrops.py*.

2.1.3 2. Computation of per-cell statistics

Per-cell statistics are computed in parallel for each channel library using *pipeline_cell_qc.py*. The pipeline computes various statistics including standard metrics such as percentage of mitochondrial reads, numbers of UMIs and numbers of genes per cell. In addition it can compute scores for custom gene sets.

Note: all per-channel matrices containing computed cell statistics are required to contain “library_id” and “barcode_id” columns. The “barcode_id” column must have the structure “umi_code-1-library_id” (e.g. AAAAAAAAAA-1-GSM0001).

Note: file names of the per-channel matrices are specified as “library_id.tsv.gz” (matrices for different analyses such as e.g. qcmetrics and scrublet scores are written to separate folders).

2.1.4 3. Cell demultiplexing [optional]

If samples have been multiplexed within channels either genetically or using hash tags a table of barcode_id -> sample_id assignments are prepared using pipeline_demux.py [not yet written].

2.1.5 4. Preparation of the cell database

The library and sample metadata, per cell statistics (and demultiplex assignments) are loaded into an sqlite database using *pipeline_celldb.py*. The pipeline creates a view called “final” which contains the qc and metadata needed for cell selection and downstream analysis.

Note: The user is required to supply a tab-separated sample metadata file (e.g. “samples.tsv”) via a path in the pipeline_celldb.yml configuration file. It should have columns for library_id, sample_id as well as any other relevant experimental metadata such as condition, genotype, age, replicate, sex etc.

2.1.6 5. Fetching of cells for downstream analysis

Cells are fetched using *pipeline_fetch_cells.py*. The user specifies the cells that they wish to retrieve from the “final” table (see step 4) via an sql statement in the pipeline_fetch_cells.yml configuration file. The pipeline will extract the cells and metadata from the original matrices and combine them into matrix matrices and anndata objects for downstream analyses.

The pipeline supports fetching of velocity information.

Note: The retrieved metadata will include a “sample_id” column. From this point onwards it is natural to think of the “sample_id” as the unit of interest. The “library_ids” remain in the metadata along with all the qc statistics to facilitate downstream investigation of batch effects and cell quality.

2.1.7 6. Integration

Alignment of samples is performed with *pipeline_integration.py*. Currently the pipeline supports harmony, bbknn and scanorama. It will produce UMAPs summarising the alignments and will compute the LISI statistic.

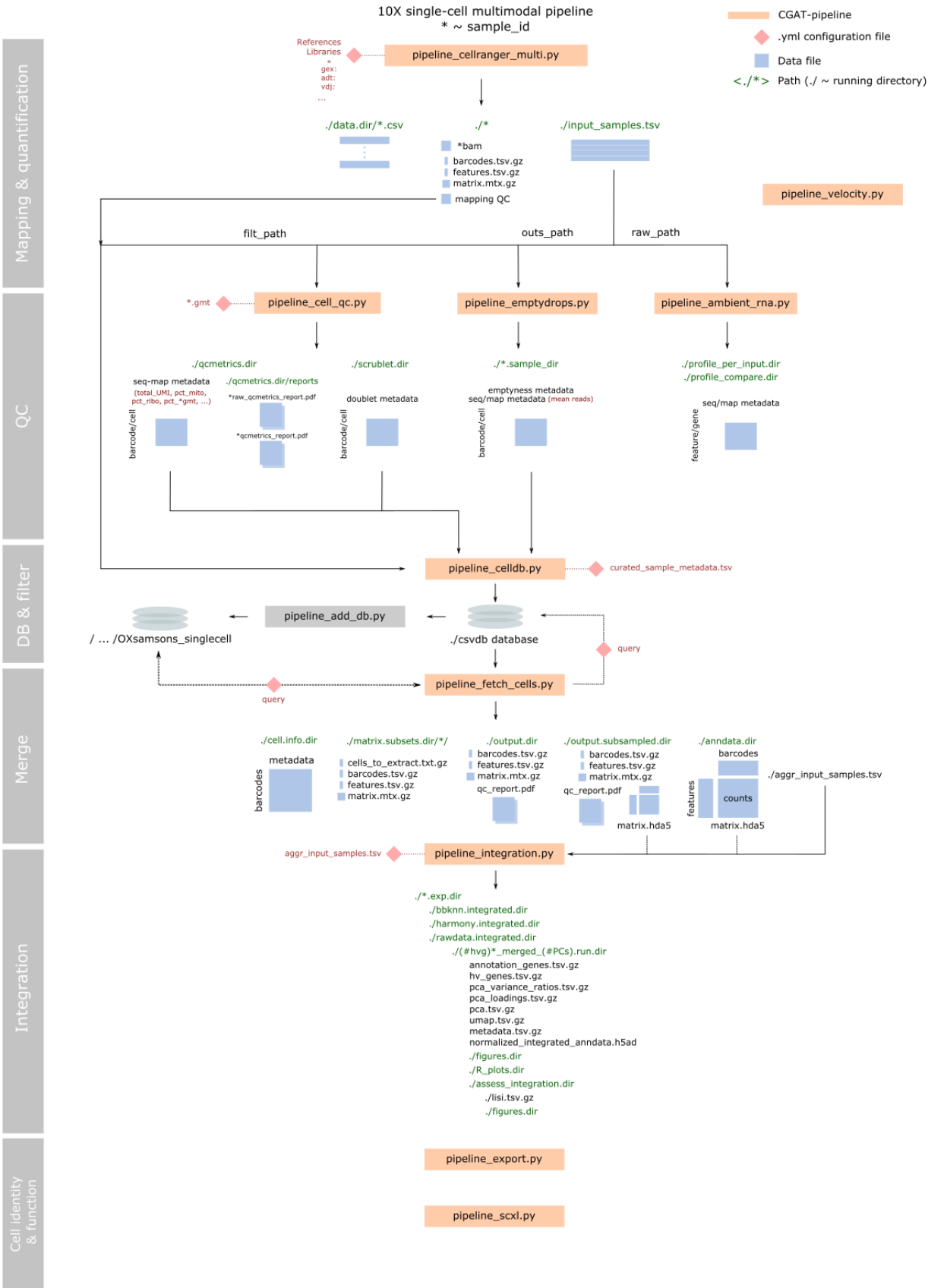
Note: The user is required to supply a tsv file (e.g. “integration.tsv”) containing the path to the pipeline_fetch_cells.py anndata object that holds the data which is to be integrated. The path to the tsv file is specified in the “pipeline_integration.yml” configuration file.

2.1.8 7. Export for seurat [optional]

The integration pipeline outputs an anndata object suitable for analysis with scanpy. A Seurat object can be prepared using *pipeline_export.py*.

2.2 Workflow Diagram

The diagram is now a little out of date with respect to configuration of the pipeline inputs but provides a useful depiction of the overall workflow.



2.3 Usage

2.3.1 Configuring and running pipelines

Run the cellhub `--help` command to view the help documentation and find available pipelines to run cellhub.

The cellhub pipelines are written using [cgat-core](#) pipelining system. For more information please see the [CGAT-core paper](#). Here we illustrate how the pipelines can be run using a toy example which is included in this repository.

Following installation, to find the available pipelines run:

```
cellhub -h
```

Next generate the configuration yml file (for the example pipeline it is empty):

```
cellhub example config -v5
```

To fully run the example cellhub pipeline run:

```
cellhub example make full -v5
```

However, it may be best to run the individual tasks of the pipeline to get a feel of what each task is doing:

```
cellhub example make exampleOriginate -v5
```

You can also run the pipeline with more advanced combinatorics by running the task:

```
cellhub example make advancedRuffus -v5
```

2.3.2 Getting Started

To get started please see the [IFNb example](#).

EXAMPLES

3.1 IFNb PBMC example

To get started create a suitable directory and cd into it.

Copy the yml, sample.tsv, integration.tsv and export.tsv configuration and metadata files for this example to the working folder:

```
cp /path/to/cellhub/examples/ifnb_pbmc/* .
```

3.1.1 1. Running Cellranger

The first step is to configure and run pipeline_cellranger_multi. We already have a pre-configured yml file so we can skip this step but the syntax is included for reference here and also for the other steps:

```
# cellhub cellranger_multi config
```

Edit the pipeline_cellranger_multi.yml file as appropriate to point to folders containing fastq files extracted from the original BAM files submitted by [Kang et. al.](#) to GEO (GSE96583). The GEO identifiers are: unstimulated (GSM2560248) and stimulated (GSM2560249). The fastqs can be extracted with the [10X bamtofastq tool](#).

We run the pipeline as follows:

```
cellhub cellranger_multi make full -v5 -p20
```

If you have not run “python setup.py devel” pipelines can instead be launched directly. In this case the equivalent command would be:

```
python path/to/cellhub-devel/cellhub/pipeline_cellranger_multi.py make full -v5 -p20 -  
↪-pipeline-log=pipeline_cellranger_multi.py
```

Note: when launching pipelines directly if the “-pipeline-log” parameter is not specified the log file will be written to “pipeline.log”.

3.1.2 2. Running the cell qc pipeline

Next we run the cell qc pipeline:

```
# cellhub cell_qc config
cellhub cell_qc make full -v5 -p20
```

3.1.3 3. Running emptydrops and investigating ambient RNA (optional)

If desired we can run emptydrops:

```
# cellhub emptydrops config
cellhub emptydrops make full -v5 -p20
```

And investigate the ambient rna:

```
# cellhub ambient_rna config
cellhub ambient_rna make full -v5 -p20
```

3.1.4 4. Loading the cell statistics into the celldb

The cell QC statistics and metadata (“samples.tsv”) are next loaded into a local sqlite database:

```
# cellhub celldb config
cellhub celldb make full -v5 -p20
```

3.1.5 5. Fetching cells for downstream analysis

We use `pipeline_fetch_cells` to retrieve the cells we want for downstream analysis. (QC thresholds and e.g. desired samples are specified in the `pipeline_fetch_cells.yml` file:

```
# cellhub fetch_cells config
cellhub fetch_cells make full -v5 -p20
```

3.1.6 6. Integration

`Pipeline_integration` supports integration of the data with `harmony`, `bbknn` and `scanorama`. In this example the location of the data is specified in the “`integration.tsv`” file as per the path given in the “`pipeline_integration.yml`” file.

```
# cellhub integration config
cellhub integration make full -v5 -p20
```

3.1.7 7. Export for downstream analysis

Finally we can export the integrated anndata object to e.g. a Seurat object for downstream analysis:

```
# cellhub export config  
cellhub export make full -v5 -p20
```

3.1.8 8. Perform downstream analysis

Downstream analysis can be performed with [pipeline_scxl.py](#). A suitable configuration file for working with the harmony aligned seurat object is provided in the `examples/ifnb_pbmc/scxl/` folder:

```
mkdir scxl.dir  
cd scxl.dir  
mkdir integrated.seurat.dir  
ln -s ../export.dir/pbmc.exp.dir/seurat_object.rds integrated.seurat.dir/begin.rds  
cp /path/to/cellhub/examples/ifnb_pbmc/pipeline_scxl/pipeline.yml .  
python /path/to/tenx/pipelines/pipeline_scxl.py make full -v5 -p200
```

CODING GUIDELINES

4.1 Coding Guidelines

4.1.1 Repository layout

Table 1: Repository layout

| Folder | Contents |
|---------------|---|
| cellhub | The cellhub Python module which contains the set of CGAT-core pipelines |
| cellhub/tasks | The cellhub tasks submodule which contains helper functions and pipeline task definitions |
| Python | Python worker scripts that are executed by pipeline tasks |
| R | R worker scripts that are executed by pipeline tasks |
| docsrc | The documentation source files in restructured text format for compilation with sphinx |
| docs | The compiled documentation |
| examples | Configuration files for example datasets |
| conda | Conda environment, TBD |
| tests | TBD |

4.1.2 Coding style

Currently we are working to improve and standardise the coding style:

- Python code should be [pep8](#) compliant. Compliance checks will be enforced soon.
- R code should follow the [tidyverse style guide](#). Please do not use right-hand assignment.
- Arguments to Python scripts should be parsed with argparse.
- Arguments to R scripts should be parsed with optparse or supplied via yaml files.
- Logging in Python scripts should be performed with the standard library “logging” module.
- Logging in R scripts should be performed with the “futile.logger” library.
- If you need to write to stdout from R scripts use message() or warning(). Do not use print() or cat().

4.1.3 Writing pipelines

The pipelines live in the “cellhub” python module.

Auxiliary task functions live in the “cellhub/task” python sub-module.

Note: Tasks of more than a few lines should be abstracted into appropriately named sub-modules.

In the notes below “xxx” denotes the name of a pipeline such as e.g. “cell_qc”.

1. Paths must never be hardcoded in the pipelines - rather they must be read from the yaml files.
2. Yaml configuration files should be named pipeline_xxx.yml
3. The output of individual pipelines should be written to a subfolder name “xxx.dir” to keep the root directory clean (it should only contain these directories and the yaml configuration files!).
4. We are documenting the pipelines using the sphinx “autodocs” module so please maintain informative rst doc-strings.

4.1.4 Yaml configuration file naming

The cgat-core system only supports configuration files name “pipeline.yml”.

We work around this by overriding the cgat-core functionality using a helper function in cellhub.tasks.control as follows:

```
import Pipeline as P
import cellhub.tasks.control as C

# Override function to collect config files
P.control.write_config_files = C.write_config_files
```

Default yaml files must be located at the path pipelines/pipeline_xxx/pipeline_xxx.yml

4.1.5 Writing and compiling the documentation

The source files for the documentation are found in:

```
docsrc
```

The documentation source files for the pipelines can be found in:

```
docsrc/pipelines
```

To build the documentation cd to the docsrc folder and run:

```
make github
```

This will build the documentation and copy the latex output to the “docs” folder. You then need to cd to the “docs” folder and run:

```
make
```

To compile the pdf.

When the repo is made public we will switch to using html documentation on readthedocs. Unfortunately there is no straightforward solution for private html hosting.

PIPELINE DOCUMENTATION

5.1 Pipeline ambient rna

5.1.1 Overview

This pipeline performs the following steps: * Analyse the ambient RNA profile in each input (eg. channel's or library's raw cellrange matrices) * Compare ambient RNA profiles across inputs

Configuration

The pipeline requires a configured `pipeline.yml` file. Default configuration files can be generated by executing:

```
python <srcdir>/pipeline_ambient_rna.py config
```

Input files

An tsv file called 'input_libraries.tsv' is required. This file must have column names as explained below. Must not include row names. Add as many rows as input channels/libraries for analysis.

This file must have the following columns:

- `library_id` - name used throughout. This could be the `channel_pool` id eg. A1
- `raw_path` - path to the `raw_matrix` folder from cellranger count
- `exp_batch` - might or might not be useful. If not used, fill with "1"
- `channel_id` - might or might not be useful. If not used, fill with "1"
- `seq_batch` - might or might not be useful. If not used, fill with "1"
- (optional) `blacklist` - path to a file with `cell_ids` to blacklist

You can add any other columns as required, for example `pool_id`

Dependencies

This pipeline requires: * cgat-core: <https://github.com/cgat-developers/cgat-core> * R dependencies required in the r scripts

5.1.2 Pipeline output

The pipeline returns: * per-input html report and tables saved in a 'profile_per_input' folder * ambient rna comparison across inputs saved in a 'profile_compare' folder

5.1.3 Code

```
cellhub.pipeline_ambient_rna.checkInputs (outfile)
    Check that input_libraries.tsv exists and the path given in the file is a valid directorys.

cellhub.pipeline_ambient_rna.genClusterJobs ()
    Generate cluster jobs for each library

cellhub.pipeline_ambient_rna.prepFolders (infile, outfile)
    Prepare folder structure for libraries

cellhub.pipeline_ambient_rna.ambient_rna_per_input (infile, outfile)
    Explore count and gene expression profiles of ambient RNA droplets per input - The output is saved in profile_per_input.dir/<input_id> - The output consists on a html report and a ambient_genes.txt.gz file - See more details of the output in the ambient_rna_per_library.R

cellhub.pipeline_ambient_rna.ambient_rna_compare (infile, outfile)
    Compare the expression of top ambient RNA genes across inputs - The output is saved in profile_compare.dir - Output includes and html report and a ambient_rna_profile.tsv - See more details of the output in the ambient_rna_compare.R

cellhub.pipeline_ambient_rna.plot (infile, outfile)
    Draw the pipeline flowchart

cellhub.pipeline_ambient_rna.full ()
    Run the full pipeline.
```

5.2 Pipeline Cellranger Multi

5.2.1 Overview

This pipeline performs the following functions:

- Alignment and quantitation (using cellranger count or cellranger multi)

5.2.2 Usage

See *Installation* and *Usage* on general information how to use CGAT pipelines.

Configuration

The pipeline requires a configured `pipeline_cellranger_multi.yml` file.

Default configuration files can be generated by executing:

```
python <srcdir>/pipeline_cellranger_multi.py config
```

Dependencies

This pipeline requires: * `cgat-core`: <https://github.com/cgat-developers/cgat-core> * `cellranger`: <https://support.10xgenomics.com/single-cell-gene-expression/>

5.2.3 Pipeline output

The pipeline returns: * the output of cellranger multi

5.2.4 Code

```
cellhub.pipeline_cellranger_multi.taskSummary(infile, outfile)
```

Make a summary of optional tasks that will be run

```
cellhub.pipeline_cellranger_multi.makeConfig(outfile)
```

Read parameters from yml file for the whole experiment and save config files as csv.

```
cellhub.pipeline_cellranger_multi.cellrangerMulti(infile, outfile)
```

Execute the cellranger multi pipeline for first sample.

```
cellhub.pipeline_cellranger_multi.full()
```

Run the full pipeline.

5.3 Pipeline Cell QC

5.3.1 Overview

This pipeline performs the following steps:

- Calculates per-cell QC metrics: `ngenes`, `total_UMI`, `pct_mitochondrial`, `pct_ribosomal`, `pct_immunoglobulin`, `pct_hemoglobin`, and any specified geneset percentage
- Runs scrublet to calculate per-cell doublet score

Configuration

The pipeline requires a configured `pipeline.yml` file. Default configuration files can be generated by executing:

```
python <srcdir>/pipeline_cell_qc.py config
```

Input files

A tsv file called 'libraries.tsv' is required. This file must have column names as explained below. Must not include row names. Add as many rows as input channels/libraries for analysis. This file must have the following columns: * `library_id` - name used throughout. This could be the `channel_pool` id eg. A1 * `path` - path to the `filtered_matrix` folder from cellranger count

Dependencies

This pipeline requires: * `cgat-core`: <https://github.com/cgat-developers/cgat-core> * R dependencies required in the `r` scripts

5.3.2 Pipeline output

The pipeline returns: * `qcmetrics.dir` folder with per-input `qcmetrics.tsv.gz` table * `scrublet.dir` folder with per-input `scrublet.tsv.gz` table

5.3.3 Code

```
cellhub.pipeline_cell_qc.checkInputs (outfile)
```

Check that the `input_libraries` file exists and the path given in the file is a valid directory.

```
cellhub.pipeline_cell_qc.qc_metrics_jobs ()
```

Generate cluster jobs for each library

```
cellhub.pipeline_cell_qc.calculate_qc_metrics (infile, outfile)
```

This task will run `R/calculate_qc_metrics.R`. It uses the `input_libraries.tsv` to read the path to the cellranger directory for each input. Output: creates a `cell.qc.dir` folder and a `library_qcmetrics.tsv.gz` table per library/channel. For additional input files check the `calculate_qc_metrics.pipeline.yml` sections: - Calculate the percentage of UMIs for genesets provided - Label barcodes as `True/False` based on whether they are part or not of a set of lists of barcodes provided

```
cellhub.pipeline_cell_qc.qc_reports_jobs ()
```

Generate cluster jobs for each library

```
cellhub.pipeline_cell_qc.build_qc_reports (infile, outfile)
```

This task will run `R/build_qc_mapping_report.R`. It expects three files in the input directory: `barcodes.tsv.gz`, `features.tsv.gz`, and `matrix.mtx.gz`. Output: creates a `library_qcmetrics_report.pdf` table per input folder

```
cellhub.pipeline_cell_qc.qc_doublet_scoring_jobs ()
```

Generate cluster jobs for each library

```
cellhub.pipeline_cell_qc.run_scrublet (infile, outfile)
```

This task will run `python/run_scrublet.py`. It uses the `input_libraries.tsv` to read the path to the cellranger directory for each input. Output: creates a `scrublet.dir` folder and a `library_scrublet.tsv.gz` table per library/channel. It also creates a doublet score histogram and a double score umap for each library/channel. Check the `scrublet` section in the `pipeline.yml` to specify other parameters

```
cellhub.pipeline_cell_gc.plot(infile, outfile)
    Draw the pipeline flowchart
cellhub.pipeline_cell_gc.full()
    Run the full pipeline.
```

5.4 Pipeline Emptydrops

5.4.1 Overview

This pipeline performs the following task:

- run emptydrops on the raw output of cellranger

5.4.2 Usage

See *Installation* and *Usage* on general information how to use CGAT pipelines.

Configuration

The pipeline requires a configured `pipeline.yml` file.

Default configuration files can be generated by executing:

```
python <srcdir>/pipeline_emptydrops.py config
```

Input files

The pipeline is run from the cellranger count output (`raw_feature_bc_matrix` folder).

The pipeline expects a tsv file containing a column named `path` and a column named `sample_id`.

‘raw path’ should contain the path to each cellranger path to `raw_feature_bc_matrix`. ‘sample_id’ is the desired name for each sample (output folder will be named like this).

Dependencies

This pipeline requires: * cgat-core: <https://github.com/cgat-developers/cgat-core> * R + packages

5.4.3 Pipeline output

The pipeline returns: A list of barcodes passing emptydrops cell identification and a table with barcode ranks including all barcodes (this can be used for knee plots).

5.4.4 Code

```
cellhub.pipeline_emptydrops.checkInputs (outfile)
    Check that input_libraries.tsv exists and the path given in the file is a valid directorys.

cellhub.pipeline_emptydrops.genClusterJobs ()
    Generate cluster jobs for each library

cellhub.pipeline_emptydrops.runEmptyDrops (infile, outfile)
    Run Rscript to run EmptyDrops on each library

cellhub.pipeline_emptydrops.genClusterJobsMeans ()
    Generate cluster jobs for each library

cellhub.pipeline_emptydrops.calculateMeanReadsPerCell (infile, outfile)
    Calculate the mean reads per cell

cellhub.pipeline_emptydrops.full ()
    Run the full pipeline.
```

5.5 Pipeline Velocity

5.5.1 Overview

This pipeline performs the following steps:

- sort bam file by cell barcode
- estimate intronic and exonic reads using velocity (on selected barcodes)

5.5.2 Usage

See [Installation](#) and [Usage](#) on general information how to use CGAT pipelines.

Configuration

The pipeline requires a configured `pipeline_velocity.yml` file.

Default configuration files can be generated by executing:

```
python <srcdir>/pipeline_velocity.py config
```

Input files

The pipeline is run from bam files generated by cellranger count.

The pipeline expects a tsv file containing the path to each cellranger bam file (path) and the respective sample_id for each sample. In addition a list of barcodes is required, this could be the filtered barcodes from cellranger or a custom input (can be gzipped file). Any further metadata can be added to the file. The required columns are sample_id, barcodes and path.

Dependencies

This pipeline requires: * cgat-core: <https://github.com/cgat-developers/cgat-core> * samtools * velocity

5.5.3 Pipeline output

The pipeline returns: * a loom file with intronic and exonic reads for use in scvelo analysis

5.5.4 Code

```
cellhub.pipeline_velocity.checkInputs (outfile)
```

Check that input_samples.tsv exists and the path given in the file is a valid directory.

```
cellhub.pipeline_velocity.genClusterJobs ()
```

Generate cluster jobs for each sample

```
cellhub.pipeline_velocity.sortBam (infile, outfile)
```

Sort bam file by cell barcodes

```
cellhub.pipeline_velocity.runVelocityto (infile, outfile)
```

Run velocity on barcode-sorted bam file. This task writes a loom file into the pipeline-run directory for each sample.

```
cellhub.pipeline_velocity.full ()
```

Run the full pipeline.

5.6 Pipeline celldb

5.6.1 Overview

This pipeline uploads the outputs from the upstream single-cell preprocessing steps into a SQLite database.

5.6.2 Usage

See *Installation* and *Usage* for general information on how to use cgat pipelines.

Configuration

The pipeline requires a configured `pipeline.yml` file.

Default configuration files can be generated by executing: `cellhub celldb config`

Input files

The pipeline requires the output of the pipelines: >> pipeline_cellranger.py : sample/10X-chip-channel x design-metadata >> pipeline_qc_metrics.py : barcode/cell x sequencing + mapping metadata >> pipeline_ambient_rna.py : gene/feature x sequencing + mapping metadata

pipeline generates a tsv configured file.

Dependencies

5.6.3 Pipeline output

The pipeline returns an SQLite populated database of metadata and quality features that aid the selection of ‘good’ cells from ‘bad’ cells.

Currently the following tables are generated: * metadata

5.6.4 Code

```
cellhub.pipeline_cellldb.connect()  
    connect to database. Use this method to connect to additional databases. Returns a database connection.  
cellhub.pipeline_cellldb.load_samples(outfile)  
    load the sample metadata table  
cellhub.pipeline_cellldb.load_libraries(outfile)  
    load the library metadata table  
cellhub.pipeline_cellldb.load_cellranger_stats(outfile)  
    load metadata of mapping data into database  
cellhub.pipeline_cellldb.load_gex_qcmetrics(outfile)  
    load the gex qcmetrics into the database  
cellhub.pipeline_cellldb.load_gex_scrublet(outfile)  
    load the scrublet scores into database  
cellhub.pipeline_cellldb.final(outfile)
```

5.7 Pipeline fetch cells

5.7.1 Overview

This pipeline fetches a given set of cells from market matrices or loom files into a single market matrix file.

5.7.2 Usage

See *Installation* and *Usage* on general information how to use CGAT pipelines.

Configuration

The pipeline requires a configured `pipeline_fetch_cells.yml` file.

Default configuration files can be generated by executing:

```
python <srcdir>/pipeline_fetch_cells.py config
```

Input files

There are two inputs, both specified in the `pipeline.yml` file.

- (1) A map of cell barcodes to matrix identifiers

This file should be a gzipped tsv file with two columns. Column 1 should have the title “barcode” and contain the cell barcodes. Column 2 should have the title “matrix_id” and contain a matrix identifier

```
barcode matrix_id ACCCATCG channel_1 ATTCATCG channel_1 AGGCATCG channel_2 TCCCATCG channel_2
gCCCATCG channel_3
```

- (2) A table containing the matrix identifiers, locations and types.

This file should be a tsv file with three columns:

```
matrix_id matrix_dir matrix_type channel_1 /full/path/channel_1_matrix mm channel_2 /full/path/channel_2_matrix
mm channel_3 /full/path/channel_3_matrix mm
```

The matrix type should be specified as either “mm” for market matrix format or “loom” for the loom file format.

Dependencies

This pipeline requires:

5.7.3 Pipeline output

The pipeline outputs a folder containing a single market matrix that contains the requested cells.

```
cellhub.pipeline_fetch_cells.fetch_cell_table (infile, outfile)
```

Fetch the table of the user’s desired cells from the database effectively, cell-metadata tsv table.

```
cellhub.pipeline_fetch_cells.matrix_subset_jobs ()
```

Generate a list of subsets from the cell metadata

```
cellhub.pipeline_fetch_cells.setupSubsetJobs (infile, outfile)
```

Setup the folders for the subsetting jobs

```
cellhub.pipeline_fetch_cells.cellSubsets (infile, outfile)
```

Extract a given subset of cells from a matrix if more than a feature modality, the `extrac_cells.R` function will generate modality specific datasets

```
cellhub.pipeline_fetch_cells.mergeSubsets (infile, outfile)
```

merge the market matrix files into a single matrix

```
cellhub.pipeline_fetch_cells.build_qc_reports (infile, outfile)
```

This task will run R/build_qc_mapping_report.R. It expects three files in the input directory barcodes.tsv.gz, features.tsv.gz, and matrix.mtx.gz. Output: creates a _qcmetrics_report.pdf

```
cellhub.pipeline_fetch_cells.transcriptDownsample (infile, outfile)
```

Down-sample transcripts given a cell-metadata variable

```
cellhub.pipeline_fetch_cells.build_subsampled_qc_reports (infile, outfile)
```

This task will run R/build_qc_mapping_report.R. It expects three files in the input directory barcodes.tsv.gz, features.tsv.gz, and matrix.mtx.gz. Output: creates a _qcmetrics_report.pdf

```
cellhub.pipeline_fetch_cells.exportSubAnnData (infiles, outfile)
```

Export a h5ad anndata matrix for downstream analysis with scanpy

```
cellhub.pipeline_fetch_cells.exportAnnData (infiles, outfile)
```

Export a h5ad anndata matrix for downstream analysis with scanpy

5.8 Pipeline Integration

5.8.1 Overview

This pipeline combines different integration methods, mainly using Satija lab's Seurat package (<http://satijalab.org/seurat/>).

For key parameters a range of choices can be specified. The pipeline will generate one report for each sample or experiment. All integration tasks and parameter combinations will be run in parallel on an HPC cluster.

The pipeline also assesses the integration by using metrics from the Seurat paper, the iLISI (harmony), the k-bet package and (optional) a similar entropy metric to what is suggested in the conos alignment method.

5.8.2 Usage

See *Installation* and *Usage* on general information how to use CGAT pipelines.

Configuration

The pipeline requires a configured `pipeline.yml` file.

Default configuration files can be generated by executing:

```
python <dropdir>/pipeline_integration.py config
```

Input files

The pipeline takes as input a file named `input_samples.tsv`. This file has to contain a column named `sample_id` representing the name for a sample or aggregated experiment. The other required column is 'path'. This has to be the path to the directory containing the aggregated matrix, metadata, barcode and features file for the respective sample (e.g. dropflow qc output)

```
sample_id path pbmc /path/to/matrix/
```

```
$ ls /path/to/matrix/ barcodes.tsv.gz features.tsv.gz matrix.mtx.gz metadata.tsv.gz
```


Dependencies

This pipeline requires:

- cgat-core: <https://github.com/cgat-developers/cgat-core>
- R & various packages.

5.8.3 Pipeline output

For each sample and each combination of parameters the following is generated within a folder for each tool+parameter:

- UMAP colored by the metadata column used for integration
- output of kbet, Seurats integration methods and harmony's iLISI in folder assess.integration.dir
- folder summary.plots.dir with summaries of each metric above
- folder summary.plots.dir also contains a csv file with the metrics
- (optional) within each folder with different cluster resolutions, a folder with entropy plots and UMAP with cluster assignments

`cellhub.pipeline_integration.checkInputs (outfile)`

Check that input_samples.tsv exists and the path given in the file exists. Then make one folder for each experiments called *.exp.dir

`cellhub.pipeline_integration.genClusterJobs ()`

Generate cluster jobs with all paramter combinations.

`cellhub.pipeline_integration.prepFolders (infile, outfile)`

Task to prepare folders for integration

`cellhub.pipeline_integration.runScanpyIntegration (infile, outfile)`

Run scanpy normalization, hv genes and harmony on the data

`cellhub.pipeline_integration.runScanpyUMAP (infile, outfile)`

Run scanpy UMAP and make plots

`cellhub.pipeline_integration.plotUMAP (infile, outfile)`

Plot UMAP with different variables

`cellhub.pipeline_integration.genJobsSummary ()`

Job generator for summary jobs

`cellhub.pipeline_integration.summariseUMAP (infile, outfile)`

Summarise UMAP from different methods

`cellhub.pipeline_integration.runLISIPy (infile, outfile)`

Assess the integration using iLISI (lisi on batch/dataset). Use the python implementation as part of the harmony package

`cellhub.pipeline_integration.genJobsLISI ()`

Job generator for LISI summary jobs

`cellhub.pipeline_integration.summariseLISI (infile, outfile)`

summarise the results from iLISI analysis

`cellhub.pipeline_integration.plot (infile, outfile)`

Draw the pipeline flowchart

5.9 Pipeline Export

5.9.1 Overview

This pipeline performs the following task:

- convert anndata to seurat object

5.9.2 Usage

See *Installation* and *Usage* on general information how to use CGAT pipelines.

Configuration

The pipeline requires a configured `pipeline.yml` file.

Default configuration files can be generated by executing:

```
python <srcdir>/pipeline_emptydrops.py config
```

Input files

The pipeline is run from the anndata object created by `pipeline_integration.py`. A sample name, the path to the anndata object (path) and (optional) path to the folder with the original matrix data (matrixdir) are specified in `input_samples.tsv`. If matrixdir is specified then market matrix format input for the full dataset are expected. The alternative option is to supply the path to a anndata object instead of the matrixdir. The option `use_full_anndata` in `pipeline.yml` needs to be set if anndata should be used.

The column names in `input_samples.tsv` need to be `sample_id`, `path` and either `matrixdir` or `anndata`. The dim reduction to include in the output seurat object is specified in the `pipeline.yml`.

Dependencies

This pipeline requires: * cgat-core: <https://github.com/cgat-developers/cgat-core> * python - scanpy, anndata * R - Seurat

5.9.3 Pipeline output

The pipeline returns: A seurat object.

5.9.4 Code

```
cellhub.pipeline_export.checkInputs (outfile)
    Check that input_samples.tsv exists and the path given in the file is a valid directorys.

cellhub.pipeline_export.genClusterJobs ()
    Generate cluster jobs for each sample

cellhub.pipeline_export.exportFromAnndata (infile, outfile)
    Run python script to extract data from anndata object
```

```
cellhub.pipeline_export.createSeuratObject (infile, outfile)  
    Run R script to create seurat object  
cellhub.pipeline_export.full()  
    Run the full pipeline.
```

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

C

`cellhub.pipeline_ambient_rna`, [13](#)
`cellhub.pipeline_cell_qc`, [15](#)
`cellhub.pipeline_cellldb`, [19](#)
`cellhub.pipeline_cellranger_multi`, [14](#)
`cellhub.pipeline_emptydrops`, [17](#)
`cellhub.pipeline_export`, [23](#)
`cellhub.pipeline_fetch_cells`, [20](#)
`cellhub.pipeline_integration`, [22](#)
`cellhub.pipeline_velocity`, [18](#)

A

`ambient_rna_compare()` (in module *cellhub.pipeline_ambient_rna*), 14
`ambient_rna_per_input()` (in module *cellhub.pipeline_ambient_rna*), 14

B

`build_qc_reports()` (in module *cellhub.pipeline_cell_qc*), 16
`build_qc_reports()` (in module *cellhub.pipeline_fetch_cells*), 21
`build_subsampled_qc_reports()` (in module *cellhub.pipeline_fetch_cells*), 22

C

`calculate_qc_metrics()` (in module *cellhub.pipeline_cell_qc*), 16
`calculateMeanReadsPerCell()` (in module *cellhub.pipeline_emptydrops*), 18
`cellhub.pipeline_ambient_rna` module, 13
`cellhub.pipeline_cell_qc` module, 15
`cellhub.pipeline_cellldb` module, 19
`cellhub.pipeline_cellranger_multi` module, 14
`cellhub.pipeline_emptydrops` module, 17
`cellhub.pipeline_export` module, 23
`cellhub.pipeline_fetch_cells` module, 20
`cellhub.pipeline_integration` module, 22
`cellhub.pipeline_velocity` module, 18
`cellrangerMulti()` (in module *cellhub.pipeline_cellranger_multi*), 15
`cellSubsets()` (in module *cellhub.pipeline_fetch_cells*), 21

`checkInputs()` (in module *cellhub.pipeline_ambient_rna*), 14
`checkInputs()` (in module *cellhub.pipeline_cell_qc*), 16
`checkInputs()` (in module *cellhub.pipeline_emptydrops*), 18
`checkInputs()` (in module *cellhub.pipeline_export*), 24
`checkInputs()` (in module *cellhub.pipeline_integration*), 23
`checkInputs()` (in module *cellhub.pipeline_velocity*), 19
`connect()` (in module *cellhub.pipeline_cellldb*), 20
`createSeuratObject()` (in module *cellhub.pipeline_export*), 24

E

`exportAnnData()` (in module *cellhub.pipeline_fetch_cells*), 22
`exportFromAnndata()` (in module *cellhub.pipeline_export*), 24
`exportSubAnnData()` (in module *cellhub.pipeline_fetch_cells*), 22

F

`fetch_cell_table()` (in module *cellhub.pipeline_fetch_cells*), 21
`final()` (in module *cellhub.pipeline_cellldb*), 20
`full()` (in module *cellhub.pipeline_ambient_rna*), 14
`full()` (in module *cellhub.pipeline_cell_qc*), 17
`full()` (in module *cellhub.pipeline_cellranger_multi*), 15
`full()` (in module *cellhub.pipeline_emptydrops*), 18
`full()` (in module *cellhub.pipeline_export*), 25
`full()` (in module *cellhub.pipeline_velocity*), 19

G

`genClusterJobs()` (in module *cellhub.pipeline_ambient_rna*), 14
`genClusterJobs()` (in module *cellhub.pipeline_emptydrops*), 18

[genClusterJobs\(\)](#) (in module *cellhub.pipeline_export*), 24
[genClusterJobs\(\)](#) (in module *cellhub.pipeline_integration*), 23
[genClusterJobs\(\)](#) (in module *cellhub.pipeline_velocity*), 19
[genClusterJobsMeans\(\)](#) (in module *cellhub.pipeline_emptydrops*), 18
[genJobsLISI\(\)](#) (in module *cellhub.pipeline_integration*), 23
[genJobsSummary\(\)](#) (in module *cellhub.pipeline_integration*), 23

L

[load_cellranger_stats\(\)](#) (in module *cellhub.pipeline_celldb*), 20
[load_gex_qcmetrics\(\)](#) (in module *cellhub.pipeline_celldb*), 20
[load_gex_scrublet\(\)](#) (in module *cellhub.pipeline_celldb*), 20
[load_libraries\(\)](#) (in module *cellhub.pipeline_celldb*), 20
[load_samples\(\)](#) (in module *cellhub.pipeline_celldb*), 20

M

[makeConfig\(\)](#) (in module *cellhub.pipeline_cellranger_multi*), 15
[matrix_subset_jobs\(\)](#) (in module *cellhub.pipeline_fetch_cells*), 21
[mergeSubsets\(\)](#) (in module *cellhub.pipeline_fetch_cells*), 21
 module
 [cellhub.pipeline_ambient_rna](#), 13
 [cellhub.pipeline_cell_qc](#), 15
 [cellhub.pipeline_celldb](#), 19
 [cellhub.pipeline_cellranger_multi](#), 14
 [cellhub.pipeline_emptydrops](#), 17
 [cellhub.pipeline_export](#), 23
 [cellhub.pipeline_fetch_cells](#), 20
 [cellhub.pipeline_integration](#), 22
 [cellhub.pipeline_velocity](#), 18

P

[plot\(\)](#) (in module *cellhub.pipeline_ambient_rna*), 14
[plot\(\)](#) (in module *cellhub.pipeline_cell_qc*), 16
[plot\(\)](#) (in module *cellhub.pipeline_integration*), 23
[plotUMAP\(\)](#) (in module *cellhub.pipeline_integration*), 23
[prepFolders\(\)](#) (in module *cellhub.pipeline_ambient_rna*), 14
[prepFolders\(\)](#) (in module *cellhub.pipeline_integration*), 23

Q

[qc_doublet_scoring_jobs\(\)](#) (in module *cellhub.pipeline_cell_qc*), 16
[qc_metrics_jobs\(\)](#) (in module *cellhub.pipeline_cell_qc*), 16
[qc_reports_jobs\(\)](#) (in module *cellhub.pipeline_cell_qc*), 16

R

[run_scrublet\(\)](#) (in module *cellhub.pipeline_cell_qc*), 16
[runEmptyDrops\(\)](#) (in module *cellhub.pipeline_emptydrops*), 18
[runLISIPy\(\)](#) (in module *cellhub.pipeline_integration*), 23
[runScanpyIntegration\(\)](#) (in module *cellhub.pipeline_integration*), 23
[runScanpyUMAP\(\)](#) (in module *cellhub.pipeline_integration*), 23
[runVelocityto\(\)](#) (in module *cellhub.pipeline_velocity*), 19

S

[setupSubsetJobs\(\)](#) (in module *cellhub.pipeline_fetch_cells*), 21
[sortBam\(\)](#) (in module *cellhub.pipeline_velocity*), 19
[summariseLISI\(\)](#) (in module *cellhub.pipeline_integration*), 23
[summariseUMAP\(\)](#) (in module *cellhub.pipeline_integration*), 23

T

[taskSummary\(\)](#) (in module *cellhub.pipeline_cellranger_multi*), 15
[transcriptDownsample\(\)](#) (in module *cellhub.pipeline_fetch_cells*), 22