# cellhub

## *Release 0.1*

**Sansom lab**

**May 06, 2021**

# CONTENTS

**1 Coding guidelines** 1
  1.1 CodingGuidelines . . . . . 1

**2 Installation** 3
  2.1 Installation . . . . . 3

**3 Getting started** 5
  3.1 Workflow Overview . . . . . 6
  3.2 IFNb PBMC example . . . . . 7
  3.3 1. Running cellranger . . . . . 7
  3.4 2. Running the cell qc pipeline . . . . . 7
  3.5 3. Running emptydrops (optional) . . . . . 7
  3.6 4. Loading the cell statistics into the celldb . . . . . 7
  3.7 5. Fetching cells for downstream analysis . . . . . 8
  3.8 6. Integration . . . . . 8
  3.9 7. Export for downstream analysis . . . . . 8

**4 Pipeline documentation** 9
  4.1 Pipeline ambient rna . . . . . 9
  4.2 Pipeline Cellranger Multi . . . . . 10
  4.3 Pipeline Cell QC . . . . . 11
  4.4 Pipeline Emptydrops . . . . . 13
  4.5 Pipeline Velocity . . . . . 14
  4.6 Pipeline celldb . . . . . 15
  4.7 Pipeline fetch cells . . . . . 17
  4.8 Pipeline Integration . . . . . 18
  4.9 Pipeline Export . . . . . 20

**5 Indices and tables** 23

**Python Module Index** 25

**Index** 27

CHAPTER

# ONE

# CODING GUIDELINES

## 1.1 CodingGuidelines

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

1. Yaml configuration files should be named pipeline_xxx.yml
2. 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 yml configuration files!).
3. Python code should pass pep8 checks, this will be enforced at some point in the future.
4. Pipelines should be documented using the sphinx “autodocs” module
5. For the autodocs documentation system to work pipelines should not read or write files outside of ruffus tasks (see below).

### 1.1.1 Writing pipelines

The pipelines live in the “pipelines” folder/module.

Auxillary task functions live in the “pipelines/task” folder/module.

If you need to read or write files outside of ruffus tasks, for example in generator functions it is essential to test that the script has not been imported e.g.:

```
if __name__ == "__main__":
   # read file
   yield(inputs, outputs)
else:
   # don't read file
   yield(None, None)
```

The reason for this is that the sphinx autodocs module needs to import the piplines to build the documentation and this will fail if the pipeline attempts to read or write files from/to non-existent paths when imported.

### 1.1.2 Yaml configuration files

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 pipelines/task/control.py as follows:

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

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

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

### 1.1.3 Writing documentation

The sources 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.

CHAPTER

# TWO

# INSTALLATION

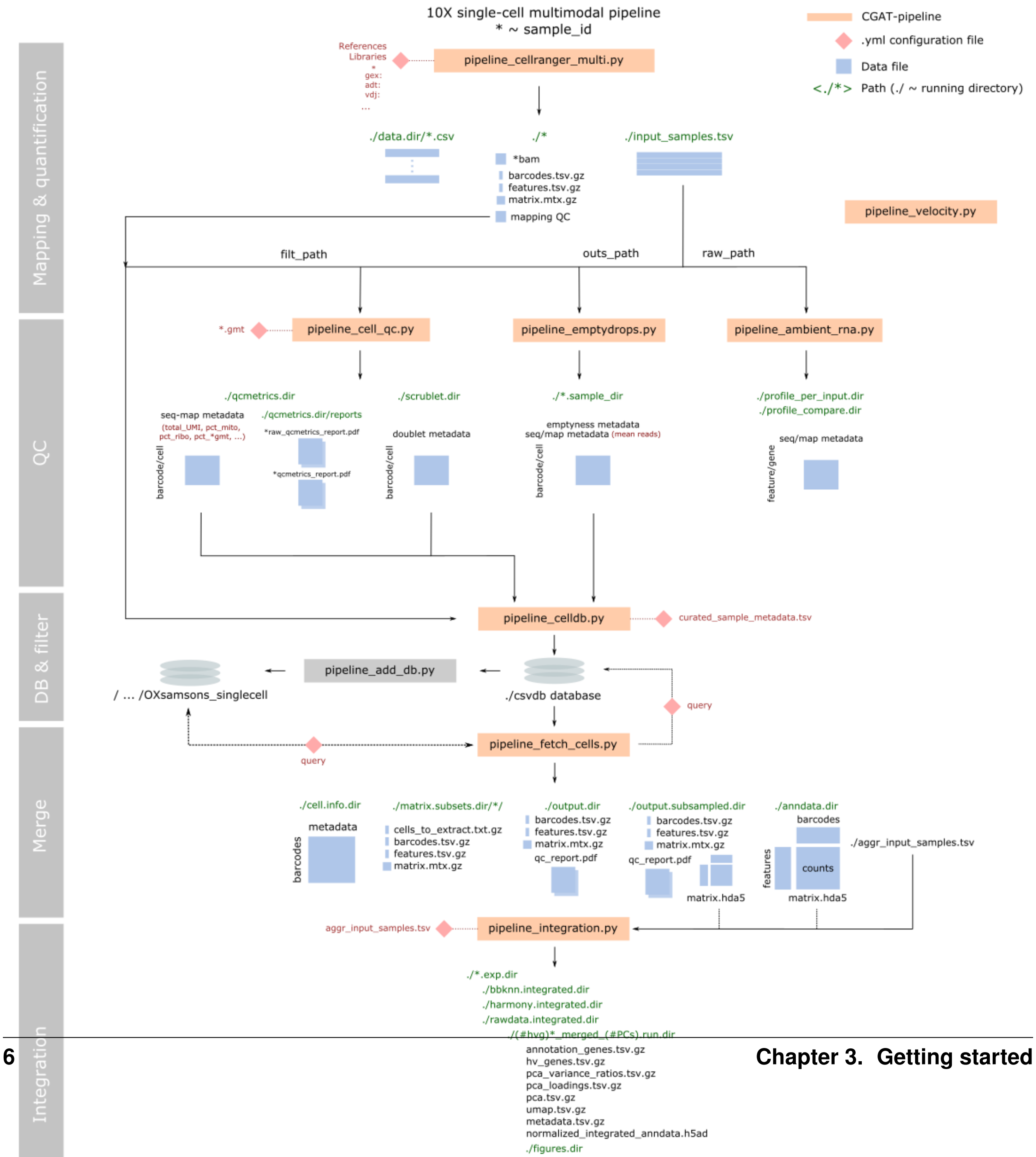
## 2.1 Installation

TBD.

# CHAPTER THREE

# GETTING STARTED

## 3.1 Workflow Overview


10X single-cell multimodal pipeline
* ~ sample_id
CGAT-pipeline
.yml configuration file
Data file
<./*> Path (./ ~ running directory)
Mapping & quantification
QC
DB & filter
Merge
Integration
References
Libraries
*
gex:
adt:
vdj:
...
pipeline_cellranger_multi.py
./data.dir/*.csv
./*
*bam
barcodes.tsv.gz
features.tsv.gz
matrix.mtx.gz
mapping QC
./input_samples.tsv
pipeline_velocity.py
filt_path
outs_path
raw_path
*.gmt
pipeline_cell_qc.py
pipeline_emptydrops.py
pipeline_ambient_rna.py
./qcmetrics.dir
seq-map metadata
(total_UMI, pct_mito, pct_ribo, pct_*gmt, ...)
./qcmetrics.dir/reports
*raw_qcmetrics_report.pdf
*qcmetrics_report.pdf
barcode/cell
./scrublet.dir
doublet metadata
barcode/cell
./*.sample_dir
emptyness metadata
seq/map metadata (mean reads)
barcode/cell
./profile_per_input.dir
./profile_compare.dir
seq/map metadata
feature/gene
pipeline_celldb.py
curated_sample_metadata.tsv
pipeline_add_db.py
/ ... /OXsamsons_singlecell
./csvdb database
query
pipeline_fetch_cells.py
query
./cell.info.dir
metadata
barcodes
./matrix.subsets.dir/*/
cells_to_extract.txt.gz
barcodes.tsv.gz
features.tsv.gz
matrix.mtx.gz
./output.dir
barcodes.tsv.gz
features.tsv.gz
matrix.mtx.gz
qc_report.pdf
./output.subsampled.dir
barcodes.tsv.gz
features.tsv.gz
matrix.mtx.gz
qc_report.pdf
matrix.hda5
./anndata.dir
barcodes
features
counts
matrix.hda5
./aggr_input_samples.tsv
aggr_input_samples.tsv
pipeline_integration.py
./*.exp.dir
./bbknn.integrated.dir
./harmony.integrated.dir
./rawdata.integrated.dir
./(#hvg)*_merged_(#PCs).run.dir
annotation_genes.tsv.gz
hv_genes.tsv.gz
pca_variance_ratios.tsv.gz
pca_loadings.tsv.gz
pca.tsv.gz
umap.tsv.gz
metadata.tsv.gz
normalized_integrated_anndata.h5ad
./figures.dir

## 3.2 IFNb PBMC example

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

Copy the sample_metadata.tsv file from the examples/ifnb_pbmc/sample_metadata.tsv [TODO!]

## 3.3 1. Running cellranger

The first step is to configure and run pipeline_cellranger_multi:

```
python /path/to/cellhub-devel/pipelines/pipeline_cellranger_multi.py config

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

## 3.4 2. Running the cell qc pipeline

Next we run the cell qc pipeline:

```
python /path/to/cellhub-devel/pipelines/pipeline_cell_qc.py config

python /path/to/cellhub-devel/pipelines/pipeline_cell_qc.py make full -v5 -p20
```

## 3.5 3. Running emptydrops (optional)

If desired we can run emptydrops:

```
python /path/to/cellhub-devel/pipelines/pipeline_emptydrops.py config

python /path/to/cellhub-devel/pipelines/pipeline_emptydrops.py make full -v5 -p20
```

## 3.6 4. Loading the cell statistics into the celldb

The cell QC statistics and metadata are next loaded into a local sqlite database:

```
python /path/to/cellhub-devel/pipelines/pipeline_celldb.py config

python /path/to/cellhub-devel/pipelines/pipeline_celldb.py make full -v5 -p20
```

## 3.7 5. Fetching cells for downstream analysis

We use pipeline_fetch_cells.py 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:

```
python /path/to/cellhub-devel/pipelines/pipeline_fetch_cells.py config

python /path/to/cellhub-devel/pipelines/pipeline_fetch_cells.py make full -v5 -p20
```

## 3.8 6. Integration

pipeline_integration.py supports integration of the data with harmony, bbknn and scanorama:

```
python /path/to/cellhub-devel/pipelines/pipeline_integration.py config

python /path/to/cellhub-devel/pipelines/pipeline_integration.py make full -v5 -p20
```

## 3.9 7. Export for downstream analysis

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

```
python /path/to/cellhub-devel/pipelines/pipeline_export.py config

python /path/to/cellhub-devel/pipelines/pipeline_export.py make full -v5 -p20
```

CHAPTER

FOUR

# PIPELINE DOCUMENTATION

## 4.1 Pipeline ambient rna

### 4.1.1 Overview

This pipeline performs the following steps: * Analyse the ambient RNA profile in each input (eg. channel's or sample'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_samples.tsv' is required. This file must have column names as explained below. Must not include row names. Add as many rows as iput channels/samples for analysis.

This file must have the following columns:

- sample_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

### 4.1.2 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

### 4.1.3 Code

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

`pipelines.pipeline_ambient_rna.`**`genClusterJobs`**`()`
Generate cluster jobs for each sample

`pipelines.pipeline_ambient_rna.`**`prepFolders`**`(`*infile*, *outfile*`)`
Prepare folder structure for samples

`pipelines.pipeline_ambient_rna.`**`ambient_rna_per_input`**`(`*infiles*, *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_sample.R

`pipelines.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

`pipelines.pipeline_ambient_rna.`**`plot`**`(`*infile*, *outfile*`)`
Draw the pipeline flowchart

`pipelines.pipeline_ambient_rna.`**`full`**`()`
Run the full pipeline.

## 4.2 Pipeline Cellranger Multi

### 4.2.1 Overview

This pipeline performs the following functions:

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

### 4.2.2 Usage

See PipelineSettingUp and PipelineRunning 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/

### 4.2.3 Pipeline output

The pipeline returns: * the output of cellranger multi

### 4.2.4 Code

`pipelines.pipeline_cellranger_multi.`**`taskSummary`**`(`*infile*, *outfile*`)`
> Make a summary of optional tasks that will be run

`pipelines.pipeline_cellranger_multi.`**`makeConfig`**`(`*outfile*`)`
> Read parameters from yml file for the whole experiment and save config files as csv.

`pipelines.pipeline_cellranger_multi.`**`cellrangerMulti`**`(`*infile*, *outfile*`)`
> Execute the cellranger multi pipleline for first sample.

`pipelines.pipeline_cellranger_multi.`**`full`**`()`
> Run the full pipeline.

## 4.3 Pipeline Cell QC

### 4.3.1 Overview

This pipeline performs the following steps: * Calculates per-cell QC metrics: ngenes, total_UMI, pct_mitochondrial,

> pct_ribosomal, pct_immunoglobin, 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 'input_samples.tsv' is required. This file must have column names as explained below. Must not include row names. Add as many rows as input channels/samples for analysis. This file must have the following columns: * sample_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

## 4.3.2 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

## 4.3.3 Code

`pipelines.pipeline_cell_qc.`**`checkInputs`**(*outfile*)

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

`pipelines.pipeline_cell_qc.`**`qc_metrics_jobs`**()

> Generate cluster jobs for each sample

`pipelines.pipeline_cell_qc.`**`calculate_qc_metrics`**(*infile*, *outfile*)

> This task will run R/calculate_qc_metrics.R, It uses the input_samples.tsv to read the path to the cellranger directory for each input Ouput: creates a qcmetrics.dir folder and a sample_qcmetrics.tsv.gz table per sample/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

`pipelines.pipeline_cell_qc.`**`qc_reports_jobs`**()

> Generate cluster jobs for each sample

`pipelines.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 Ouput: creates a sample_qcmetrics_report.pdf table per input folder

`pipelines.pipeline_cell_qc.`**`qc_doublet_scoring_jobs`**()

> Generate cluster jobs for each sample

`pipelines.pipeline_cell_qc.`**`run_scrublet`**(*infile*, *outfile*)

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

`pipelines.pipeline_cell_qc.`**`plot`**`(`*infile*, *outfile*`)`

Draw the pipeline flowchart

`pipelines.pipeline_cell_qc.`**`full`**`()`

Run the full pipeline.

# 4.4 Pipeline Emptydrops

**Author** Kathrin Jansen

**Release** $Id$

**Date** May 06, 2021

**Tags** Python

## 4.4.1 Overview

This pipeline performs the following task:

- run emptydrops on the raw output of cellranger

## 4.4.2 Usage

See PipelineSettingUp and PipelineRunning 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

### 4.4.3 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).

### 4.4.4 Code

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

`pipelines.pipeline_emptydrops.`**`genClusterJobs`**`()`
Generate cluster jobs for each sample

`pipelines.pipeline_emptydrops.`**`runEmptyDrops`**`(`*infile*, *outfile*`)`
Run Rscript to run EmptyDrops on each sample

`pipelines.pipeline_emptydrops.`**`genClusterJobsMeans`**`()`
Generate cluster jobs for each sample

`pipelines.pipeline_emptydrops.`**`calculateMeanReadsPerCell`**`(`*infile*, *outfile*`)`
Calculate the mean reads per cell

`pipelines.pipeline_emptydrops.`**`full`**`()`
Run the full pipeline.

## 4.5 Pipeline Velocity

**Author** Kathrin Jansen

**Release** $Id$

**Date** May 06, 2021

**Tags** Python

### 4.5.1 Overview

This pipeline performs the following steps:

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

### 4.5.2 Usage

See PipelineSettingUp and PipelineRunning 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 * veloctyo

### 4.5.3 Pipeline output

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

### 4.5.4 Code

`pipelines.pipeline_velocity.`**`checkInputs`**`(`*outfile*`)`
> Check that input_samples.tsv exists and the path given in the file is a valid directorys.

`pipelines.pipeline_velocity.`**`genClusterJobs`**`()`
> Generate cluster jobs for each sample

`pipelines.pipeline_velocity.`**`sortBam`**`(`*infile*, *outfile*`)`
> Sort bam file by cell barcodes

`pipelines.pipeline_velocity.`**`runVelocyto`**`(`*infile*, *outfile*`)`
> Run velocyto on barcode-sorted bam file. This task writes a loom file into the pipeline-run directory for each sample.

`pipelines.pipeline_velocity.`**`full`**`()`
> Run the full pipeline.

## 4.6 Pipeline celldb

### 4.6.1 Overview

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

### 4.6.2 Usage

See :ref:`` and :ref:`` 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

### 4.6.3 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

### 4.6.4 Code

`pipelines.pipeline_celldb.`**`connect`**`()`
connect to database. Use this method to connect to additional databases. Returns a database connection.

`pipelines.pipeline_celldb.`**`preprocess_metadata_sequencing`**`(`*infile*, *outfile*`)`
Process metadata from the pipeline_cellranger.py output [[ it contains the column sample_id ]]

`pipelines.pipeline_celldb.`**`load_metadata_sequencing`**`(`*infile*, *outfile*`)`
load metadata of sequencing data into database

`pipelines.pipeline_celldb.`**`preprocess_metadata_mapping`**`(`*infile*, *outfile*`)`
Process metadata from the pipeline_cellranger.py output [[ it contains the column sample_id ]]

`pipelines.pipeline_celldb.`**`load_metadata_mapping`**`(`*infile*, *outfile*`)`
load metadata of mapping data into database

`pipelines.pipeline_celldb.`**`process_merged_scrublet`**`(`*infiles*, *outfile*`)`
Process concatenate data together from scrublet

`pipelines.pipeline_celldb.`**`load_merged_scrublet`**`(`*infile*, *outfile*`)`
load scrublet output data into database

`pipelines.pipeline_celldb.`**`process_merged_qcmetrics`**`(`*infiles*, *outfile*`)`
Process concatenate data together from qcmetrics

`pipelines.pipeline_celldb.`**`load_merged_qcmetrics`**`(`*infile*, *outfile*`)`
load qcmetrics output data into database

`pipelines.pipeline_celldb.`**`final`**`(`*outfile*`)`

## 4.7 Pipeline fetch cells

**Author** Sansom lab

**Release** $Id$

**Date** May 06, 2021

**Tags** Python

### 4.7.1 Overview

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

### 4.7.2 Usage

See PipelineSettingUp and PipelineRunning 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:

### 4.7.3 Pipeline output

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

`pipelines.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.

`pipelines.pipeline_fetch_cells.`**`matrix_subset_jobs`**()
: Generate a list of subsets from the cell metadata

`pipelines.pipeline_fetch_cells.`**`setupSubsetJobs`**(*infile*, *outfile*)
: Setup the folders for the subsetting jobs

`pipelines.pipeline_fetch_cells.`**`cellSubsets`**(*infile*, *outfile*)
: Extract a given subset of cells from a matrix if more than a feature modadility, the extrac_cells.R function will generate modality specific datasets

`pipelines.pipeline_fetch_cells.`**`mergeSubsets`**(*infiles*, *outfile*)
: merge the market matrix files into a single matrix

`pipelines.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 Ouput: creates a _qcmetrics_report.pdf

`pipelines.pipeline_fetch_cells.`**`transcriptDownsample`**(*infile*, *outfile*)
: Down-sample transcripts given a cell-metadata variable

`pipelines.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 Ouput: creates a _qcmetrics_report.pdf

`pipelines.pipeline_fetch_cells.`**`exportSubAnnData`**(*infiles*, *outfile*)
: Export a h5ad anndata matrix for downstream analysis with scanpy

`pipelines.pipeline_fetch_cells.`**`exportAnnData`**(*infiles*, *outfile*)
: Export a h5ad anndata matrix for downstream analysis with scanpy

## 4.8 Pipeline Integration

**Author** Kathrin Jansen

**Release** $Id$

**Date** May 06, 2021

**Tags** Python

### 4.8.1 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.

### 4.8.2 Usage

See PipelineSettingUp and PipelineRunning 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.

### 4.8.3 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

`pipelines.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

`pipelines.pipeline_integration.`**`genClusterJobs`**`()`
Generate cluster jobs with all paramter combinations.

`pipelines.pipeline_integration.`**`prepFolders`**`(`*infile*, *outfile*`)`
Task to prepare folders for integration

`pipelines.pipeline_integration.`**`runScanpyIntegration`**`(`*infile*, *outfile*`)`
Run scanpy normalization, hv genes and harmonypy on the data

`pipelines.pipeline_integration.`**`runScanpyUMAP`**`(`*infile*, *outfile*`)`
Run scanpy UMAP and make plots

`pipelines.pipeline_integration.`**`plotUMAP`**`(`*infile*, *outfile*`)`
Plot UMAP with different variables

`pipelines.pipeline_integration.`**`genJobsSummary`**`()`
Job generator for summary jobs

`pipelines.pipeline_integration.`**`summariseUMAP`**`(`*infile*, *outfile*`)`
Summarise UMAP from different methods

`pipelines.pipeline_integration.`**`runLISIpy`**`(`*infile*, *outfile*`)`
Assess the integration using iLISI (lisi on batch/dataset). Use the python implementation as part of the harmonypy package

`pipelines.pipeline_integration.`**`genJobsLISI`**`()`
Job generator for LISI summary jobs

`pipelines.pipeline_integration.`**`summariseLISI`**`(`*infile*, *outfile*`)`
summarise the results from iLISI analysis

`pipelines.pipeline_integration.`**`plot`**`(`*infile*, *outfile*`)`
Draw the pipeline flowchart

## 4.9 Pipeline Export

**Author** Kathrin Jansen

**Release** $Id$

**Date** May 06, 2021

**Tags** Python

### 4.9.1 Overview

This pipeline performs the following task:

- convert anndata to seurat object

## 4.9.2 Usage

See PipelineSettingUp and PipelineRunning 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

## 4.9.3 Pipeline output

The pipeline returns: A seurat object.

## 4.9.4 Code

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

`pipelines.pipeline_export.`**`genClusterJobs`**`()`
> Generate cluster jobs for each sample

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

`pipelines.pipeline_export.`**`createSeuratObject`**`(`*infile*, *outfile*`)`
> Run R script to create seurat object

`pipelines.pipeline_export.`**`full`**`()`
> Run the full pipeline.

## CHAPTER FIVE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

p

`pipelines.pipeline_ambient_rna`, 9
`pipelines.pipeline_cell_qc`, 11
`pipelines.pipeline_celldb`, 15
`pipelines.pipeline_cellranger_multi`, 10
`pipelines.pipeline_emptydrops`, 13
`pipelines.pipeline_export`, 20
`pipelines.pipeline_fetch_cells`, 16
`pipelines.pipeline_integration`, 18
`pipelines.pipeline_velocity`, 14

# INDEX

## A

`ambient_rna_compare()` (*in module pipelines.pipeline_ambient_rna*), 10
`ambient_rna_per_input()` (*in module pipelines.pipeline_ambient_rna*), 10

## B

`build_qc_reports()` (*in module pipelines.pipeline_cell_qc*), 12
`build_qc_reports()` (*in module pipelines.pipeline_fetch_cells*), 18
`build_subsampled_qc_reports()` (*in module pipelines.pipeline_fetch_cells*), 18

## C

`calculate_qc_metrics()` (*in module pipelines.pipeline_cell_qc*), 12
`calculateMeanReadsPerCell()` (*in module pipelines.pipeline_emptydrops*), 14
`cellrangerMulti()` (*in module pipelines.pipeline_cellranger_multi*), 11
`cellSubsets()` (*in module pipelines.pipeline_fetch_cells*), 18
`checkInputs()` (*in module pipelines.pipeline_ambient_rna*), 10
`checkInputs()` (*in module pipelines.pipeline_cell_qc*), 12
`checkInputs()` (*in module pipelines.pipeline_emptydrops*), 14
`checkInputs()` (*in module pipelines.pipeline_export*), 21
`checkInputs()` (*in module pipelines.pipeline_integration*), 19
`checkInputs()` (*in module pipelines.pipeline_velocity*), 15
`connect()` (*in module pipelines.pipeline_celldb*), 16
`createSeuratObject()` (*in module pipelines.pipeline_export*), 21

## E

`exportAnnData()` (*in module pipelines.pipeline_fetch_cells*), 18
`exportFromAnndata()` (*in module pipelines.pipeline_export*), 21
`exportSubAnnData()` (*in module pipelines.pipeline_fetch_cells*), 18

## F

`fetch_cell_table()` (*in module pipelines.pipeline_fetch_cells*), 18
`final()` (*in module pipelines.pipeline_celldb*), 16
`full()` (*in module pipelines.pipeline_ambient_rna*), 10
`full()` (*in module pipelines.pipeline_cell_qc*), 13
`full()` (*in module pipelines.pipeline_cellranger_multi*), 11
`full()` (*in module pipelines.pipeline_emptydrops*), 14
`full()` (*in module pipelines.pipeline_export*), 21
`full()` (*in module pipelines.pipeline_velocity*), 15

## G

`genClusterJobs()` (*in module pipelines.pipeline_ambient_rna*), 10
`genClusterJobs()` (*in module pipelines.pipeline_emptydrops*), 14
`genClusterJobs()` (*in module pipelines.pipeline_export*), 21
`genClusterJobs()` (*in module pipelines.pipeline_integration*), 20
`genClusterJobs()` (*in module pipelines.pipeline_velocity*), 15
`genClusterJobsMeans()` (*in module pipelines.pipeline_emptydrops*), 14
`genJobsLISI()` (*in module pipelines.pipeline_integration*), 20
`genJobsSummary()` (*in module pipelines.pipeline_integration*), 20

## L

`load_merged_qcmetrics()` (*in module pipelines.pipeline_celldb*), 16
`load_merged_scrublet()` (*in module pipelines.pipeline_celldb*), 16
`load_metadata_mapping()` (*in module pipelines.pipeline_celldb*), 16

`load_metadata_sequencing()` (*in module pipelines.pipeline_celldb*), 16

M

`makeConfig()` (*in module pipelines.pipeline_cellranger_multi*), 11
`matrix_subset_jobs()` (*in module pipelines.pipeline_fetch_cells*), 18
`mergeSubsets()` (*in module pipelines.pipeline_fetch_cells*), 18
`module`
- `pipelines.pipeline_ambient_rna`, 9
- `pipelines.pipeline_cell_qc`, 11
- `pipelines.pipeline_celldb`, 15
- `pipelines.pipeline_cellranger_multi`, 10
- `pipelines.pipeline_emptydrops`, 13
- `pipelines.pipeline_export`, 20
- `pipelines.pipeline_fetch_cells`, 16
- `pipelines.pipeline_integration`, 18
- `pipelines.pipeline_velocity`, 14

P

`pipelines.pipeline_ambient_rna`
- `module`, 9

`pipelines.pipeline_cell_qc`
- `module`, 11

`pipelines.pipeline_celldb`
- `module`, 15

`pipelines.pipeline_cellranger_multi`
- `module`, 10

`pipelines.pipeline_emptydrops`
- `module`, 13

`pipelines.pipeline_export`
- `module`, 20

`pipelines.pipeline_fetch_cells`
- `module`, 16

`pipelines.pipeline_integration`
- `module`, 18

`pipelines.pipeline_velocity`
- `module`, 14

`plot()` (*in module pipelines.pipeline_ambient_rna*), 10
`plot()` (*in module pipelines.pipeline_cell_qc*), 12
`plot()` (*in module pipelines.pipeline_integration*), 20
`plotUMAP()` (*in module pipelines.pipeline_integration*), 20
`prepFolders()` (*in module pipelines.pipeline_ambient_rna*), 10
`prepFolders()` (*in module pipelines.pipeline_integration*), 20
`preprocess_metadata_mapping()` (*in module pipelines.pipeline_celldb*), 16
`preprocess_metadata_sequencing()` (*in module pipelines.pipeline_celldb*), 16
`process_merged_qcmetrics()` (*in module pipelines.pipeline_celldb*), 16
`process_merged_scrublet()` (*in module pipelines.pipeline_celldb*), 16

Q

`qc_doublet_scoring_jobs()` (*in module pipelines.pipeline_cell_qc*), 12
`qc_metrics_jobs()` (*in module pipelines.pipeline_cell_qc*), 12
`qc_reports_jobs()` (*in module pipelines.pipeline_cell_qc*), 12

R

`run_scrublet()` (*in module pipelines.pipeline_cell_qc*), 12
`runEmptyDrops()` (*in module pipelines.pipeline_emptydrops*), 14
`runLISIpy()` (*in module pipelines.pipeline_integration*), 20
`runScanpyIntegration()` (*in module pipelines.pipeline_integration*), 20
`runScanpyUMAP()` (*in module pipelines.pipeline_integration*), 20
`runVelocyto()` (*in module pipelines.pipeline_velocity*), 15

S

`setupSubsetJobs()` (*in module pipelines.pipeline_fetch_cells*), 18
`sortBam()` (*in module pipelines.pipeline_velocity*), 15
`summariseLISI()` (*in module pipelines.pipeline_integration*), 20
`summariseUMAP()` (*in module pipelines.pipeline_integration*), 20

T

`taskSummary()` (*in module pipelines.pipeline_cellranger_multi*), 11
`transcriptDownsample()` (*in module pipelines.pipeline_fetch_cells*), 18