cellhub

Release 0.1

Sansom lab

Jul 26, 2022

CONTENTS

CHAPTER

ONE

INSTALLATION

1.1 Installation

1.1.1 Dependencies

Core dependencies include:

- Cellranger (from 10X Genomics) >= 6.0
- Python3
- Various Python packages (see python/requirements.txt)
- R >= 4.0
- Various R libraries (see R/install.packages.R)
- Latex
- The provided cellhub R library

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
```

5. Install the cellhub R library:

```
R CMD INSTALL R/cellhub
```

Note: On some systems, automatic detection of the HDF5 library by the R package "hdf5r" (a dependency of loomR) is problematic. This can be worked around by explicitely passing the path to your HDF5 library h5cc or h5pcc binary, e.g.

```
install.packages("hdf5r", configure.args="--with-hdf5=/apps/eb/dev/skylake/software/
↪HDF5/1.10.6-gompi-2020a/bin/h5pcc")
```

CHAPTER

TWO

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 eight 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 QC 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 genesets.

Per-cell celltype predictions are computed in parallel for each channel library using *pipeline_singleR.py*

Note: all per-channel matrices containing computed cell statistics are required to contain "library_id" and "barcode_id" columns.

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. Initial assessment of cell quality

This is performed manally by the data analyst as it is highly dataset-specific. Per cell QC statistics can be easily retrieved from the celldb for plotting along with singleR scores from the cellhub API.

2.1.7 6. 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 market matrices and anndata objects for downstream analyses.

It is recommended to fetch cells into a new directory. By design fetching of a single dataset per-directory is supported.

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.

2.1.8 7. ADT normalization [optional]

If samples included the ADT modality, *pipeline_adt_norm.py* normalizes the antibody counts for the high-quality fetched cells in the previous step. Normalized ADT can be then used for downstream integration. The pipeline implements 3 normalization methodologies: DSB, median-based, and CLR. The user can specify the feature space.

2.1.9 8. Integration

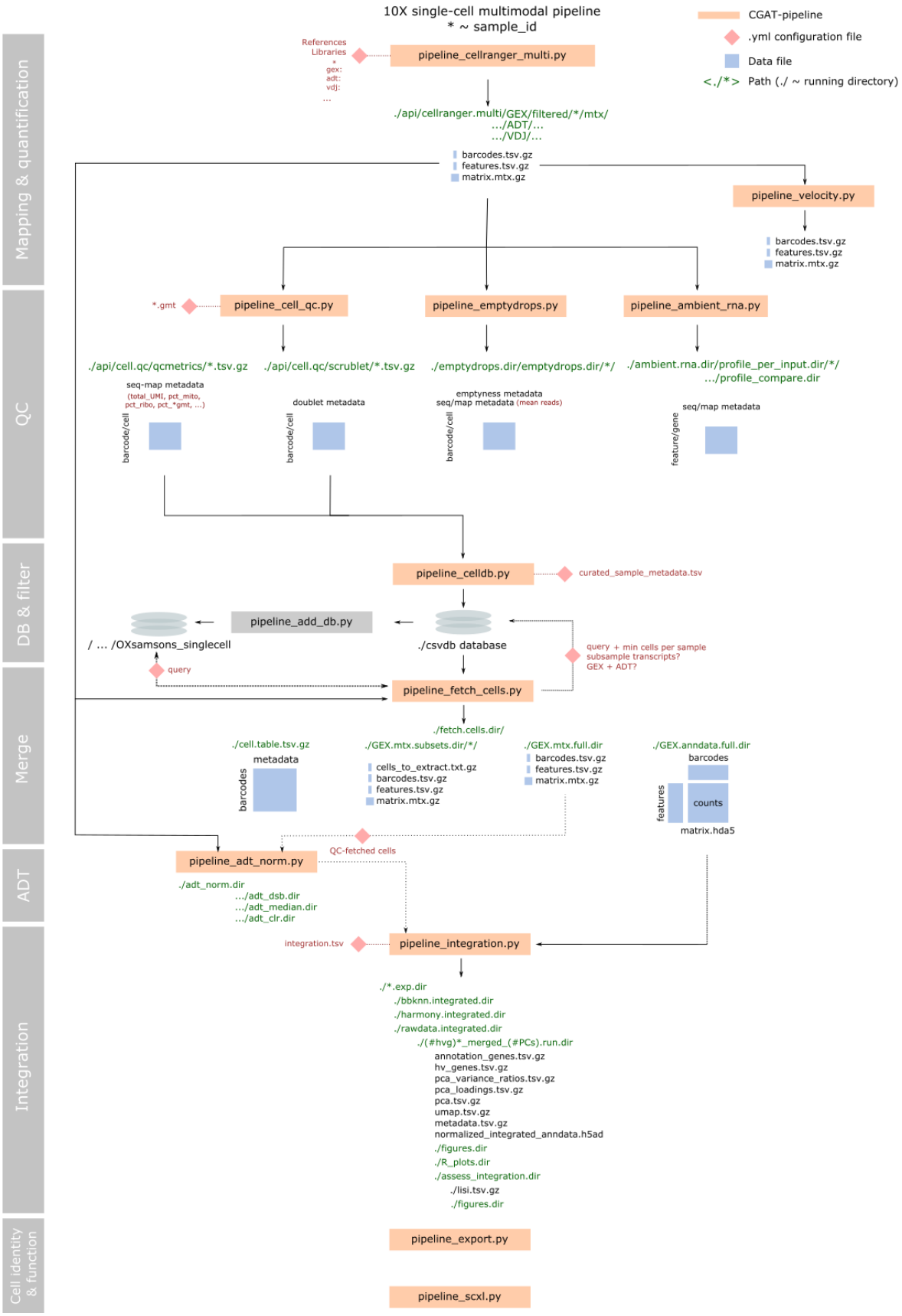
Integration of samples is performed manually by the user because it is highly dataset specific. Different integration algorithms are needed for different contexts. Strategies for HVG selection and modelling of covariates need to be considered by the data analyst on a case by case basis.

2.1.10 9. Clustering analysis

Clustering analysis is performed with *pipeline_cluster*.

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.


10X single-cell multimodal pipeline
* ~ sample_id
CGAT-pipeline
.yml configuration file
Data file
<./*> Path (./ ~ running directory)
Mapping & quantification
QC
DB & filter
Merge
ADT
Integration
Cell identity & function
References
Libraries
+
gex:
adt:
vdj:
...
pipeline_cellranger_multi.py
./api/cellranger.multi/GEX/filtered/*/mtx/
.../ADT/...
.../VDJ/...
barcodes.tsv.gz
features.tsv.gz
matrix.mtx.gz
pipeline_velocity.py
barcodes.tsv.gz
features.tsv.gz
matrix.mtx.gz
*.gmt
pipeline_cell_qc.py
pipeline_emptydrops.py
pipeline_ambient_rna.py
./api/cell.qc/qcmetrics/*.tsv.gz
./api/cell.qc/scrublet/*.tsv.gz
./emptydrops.dir/emptydrops.dir/*/
./ambient.rna.dir/profile_per_input.dir/*/
.../profile_compare.dir
seq-map metadata
(total_UMI, pct_mito,
pct_ribo, pct_*gmt, ...)
barcode/cell
doublet metadata
barcode/cell
emptyness metadata
seq/map metadata (mean reads)
barcode/cell
seq/map metadata
feature/gene
pipeline_celldb.py
curated_sample_metadata.tsv
pipeline_add_db.py
/ ... /OXsamsons_singlecell
./csvdb database
query
query + min cells per sample
subsample transcripts?
GEX + ADT?
pipeline_fetch_cells.py
./fetch.cells.dir/
./cell.table.tsv.gz
metadata
barcodes
./GEX.mtx.subsets.dir/*/
cells_to_extract.txt.gz
barcodes.tsv.gz
features.tsv.gz
matrix.mtx.gz
./GEX.mtx.full.dir
barcodes.tsv.gz
features.tsv.gz
matrix.mtx.gz
./GEX.anndata.full.dir
barcodes
features
counts
matrix.hda5
QC-fetched cells
pipeline_adt_norm.py
./adt_norm.dir
.../adt_dsb.dir
.../adt_median.dir
.../adt_clr.dir
integration.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
./R_plots.dir
./assess_integration.dir
./lisi.tsv.gz
./figures.dir
pipeline_export.py
pipeline_scxl.py

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. From more information please see the CGAT-core paper. Here we illustrate how the pipelines can be run using the cellranger_multi pipeline as an example.

Following installation, to find the available pipelines run:

```
cellhub -h
```

Next generate a configuration yml file:

```
cellhub cellranger_multi config -v5
```

To fully run the example cellhub pipeline run:

```
cellhub cellranger_multi 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. To list the pipline tasks and their current status, use the 'show' command:

```
cellhub cellranger_multi show
```

Individual tasks can then be executed by name, e.g.

```
cellhub cellranger_multi make cellrangerMulti -v5
```

Note: If any upstream tasks are out of date they will automatically be run before the named task is executed.

2.3.2 Getting Started

To get started please see the *IFNb example*.

CHAPTER

THREE

EXAMPLES

3.1 IFNb PBMC example

3.1.1 Setting up

1. Clone the example template to a local folder

Clone the folders and files for the example into a local folder.

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

This will create 3 folders:

- "cellhub" where the preprocessing pipelines will be run and where the cellhub database will be created
- "integration" where integration is to be performed.
- "cluster" where we can perform downstream analysis with "cellhub cluster".

3.1.2 Running the pre-processing pipelines and creating the database

2. 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:

```
# enter the cellhub directory

cd cellhub

# 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. Running the cell qc pipeline

Next we run the cell qc pipeline:

```
# cellhub cell_qc config

cellhub cell_qc make full -v5 -p20
```

4. 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
```

5. Run pipeline_singleR

Single R is run on all the cells so that the results are avaliable to help with QC as well as downstream analysis:

```
# cellhub singleR config

cellhub singleR make full -v5 -p20
```

As noted: *in the pipeline_singleR inputs section* the celldex references neede to be stashed before the pipeline is run.

6. 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
```

7. Run pipeline_annotation

This pipeline retrieves Ensembl and KEGG annotations needed for downstream analysis.:

```
# cellhub annotation config

cellhub annotation make full -v5 -p10
```

Please note that the specified Ensembl version should match that used for the cellranger reference trancriptome.

3.1.3 Performing cell QC

8. Assessment of cell quality

This step is left to the reader to perform manually because it needs to be carefully tailored to individual datasets.

3.1.4 Performing downstream analysis

9. Fetch cells for integration

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:

```
# It is recommended to fetch the cells in to a seperate directory for integration.
cd ../integration

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

10. Integration

Run the provided jupyter notebook to perform a basic Harmony integration of the data and to save it in the appropriate anndata format (see *in the pipeline_cluster inputs section*) is provided.

11. Clustering analysis

Cluster analysis is performed with pipeline cluster (a seperate directory is recommended for this so that multiple clustering runs can be performed as required).:

```
# change into the clustering directory
cd ../cluster.dir

# checkout the yml file
cellhub cluster config

# a suitable yml file has been provided so we can now launch the pipeline
cellhub cluster make full -v5 -p200
```

The pdf reports and excel files generated by the pipeline can be found in the "reports.dir" subfolder.

For interactive visulation, the results are provided in cellxgene format. To view the cellxgene.h5ad files, you will first need toinstall cellxgene with "pip install cellxgene". The cellxgene viewer can then be launched with:

```
# substitute "{x}" with the number integrated components used for the clustering run.
cellxgene --no-upgrade-check launch out.{x}.comps.dir/cellxgene.h5ad
```

CHAPTER

FOUR

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
cellhub/reports	The latex source files used for building the reports
Python	Python worker scripts that are executed by pipeline tasks
R/cellhub	The R cellhub library
R/scripts	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 should 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 yml configuration files!).
4. Pipelines that generate cell-level information for down-stream analysis must read their inputs from the api and register their outputs to the API, see *API*. If you need information from an upstream pipeline that is not present on the API please raise an issue.
5. We are documenting the pipelines using the sphinx "autodocs" module so please maintain informative rst doc-strings.

4.1.4 Cell barcodes

- We use cell barcodes in the format "UMI-library_id".
- Barcodes are set upstream e.g. in the market matrix files that are exposed on the API by pipeline_cellranger_multi.py.
- Downstream pipelines shold not need to manipulate barcodes. If you find barcodes being served on the API in an incorrect format please raise an issue.
- We use the field/column name "barcode_id" for the fields/columns that contains the barcodes in tables and databases.

4.1.5 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 yml files must be located at the path cellhub/yaml/pipeline_xxx.yml

4.1.6 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.

4.2 API

4.2.1 Overview

The pre-processing pipelines must define and register their outputs via a common api. The *cellhub.tasks.api module* provides the code for doing this. The api comprises of an "api" folder into which pipeline outputs are symlinked (by the "register_dataset" "api" class method).

The api provides a stable and sanitised interface from which the downstream pipelines can access the information.

4.2.2 Example

The "api" folder for the IFNB example dataset looks like this.

```
api
├── cell.qc
│   ├── qcmetrics
│   │   └── filtered
│   │       ├── GSM2560248.tsv.gz -> ../../../../cell.qc.dir/qcmetric.dir/GSM2560248.tsv.gz
│   │       ├── GSM2560249.tsv.gz -> ../../../../cell.qc.dir/qcmetric.dir/GSM2560249.tsv.gz
│   │       └── manifest.yml
│   └── scrublet
│       └── filtered
│           ├── GSM2560248.tsv.gz -> ../../../../cell.qc.dir/scrublet.dir/GSM2560248.tsv.gz
│           ├── GSM2560249.tsv.gz -> ../../../../cell.qc.dir/scrublet.dir/GSM2560249.tsv.gz
│           └── manifest.yml
└── cellranger.multi
    └── GEX
        ├── filtered
        │   ├── GSM2560248
        │   │   └── mtx
        │   │       ├── barcodes.tsv.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560248/GSM2560248/GEX/barcodes.tsv.gz
        │   │       ├── features.tsv.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560248/GSM2560248/GEX/features.tsv.gz
        │   │       ├── manifest.yml
        │   │       └── matrix.mtx.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560248/GSM2560248/GEX/matrix.mtx.gz
        │   └── GSM2560249
        │       └── mtx
        │           ├── barcodes.tsv.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560249/GSM2560249/GEX/barcodes.tsv.gz
        │           ├── features.tsv.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560249/GSM2560249/GEX/features.tsv.gz
        │           ├── manifest.yml
        │           └── matrix.mtx.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560249/GSM2560249/GEX/matrix.mtx.gz
        └── unfiltered
            ├── GSM2560248
            │   └── mtx
            │       ├── barcodes.tsv.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560248/unfiltered/GEX/barcodes.tsv.gz
            │       ├── features.tsv.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560248/unfiltered/GEX/features.tsv.gz
            │       ├── manifest.yml
            │       └── matrix.mtx.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560248/unfiltered/GEX/matrix.mtx.gz
            └── GSM2560249
                └── mtx
                    ├── barcodes.tsv.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560249/unfiltered/GEX/barcodes.tsv.gz
                    ├── features.tsv.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560249/unfiltered/GEX/features.tsv.gz
                    ├── manifest.yml
                    └── matrix.mtx.gz -> ../../../../../../cellranger.multi.dir/out.dir/GSM2560249/unfiltered/GEX/matrix.mtx.gz
```

4.2.3 Usage

Please see the *cellhub.tasks.api module documentation* for more details.

..note:: If you are writing a pipeline and information that you need from an upstream pipeline has not been exposed on the api please raise an issue.

CHAPTER

FIVE

TASKS MODULE DOCUMENTATION

CHAPTER

SIX

PIPELINE DOCUMENTATION

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search