lts$_{w}ork flows_{s}m_{s}crnaseq Documentation

Release 0.3.5+12.g9c99613.dirty

Per Unneberg

Oct 05, 2018

Contents

single-cell RNA sequencing snakemake workflow

- Free software: GNU General Public License v3

CHAPTER 1

Quickstart

```
$ conda create -n py2.7 python=2.7 rpkmforgenes=1.0.1 rseqc=2.6.4
$ conda create -n lts-workflows-sm-scrnaseq -c scilifelab-lts lts-workflows-sm-
↪scrnaseq
$ lts_workflows_sm_scrnaseq
$ lts_workflows_sm_scrnaseq -l
$ lts_workflows_sm_scrnaseq all -d /path/to/workdir --configfile config.yaml
$ lts_workflows_sm_scrnaseq --use-conda all -d /path/to/workdir --configfile config.
↪yaml
```

```
$ docker pull scilifelablts/lts-workflows-sm-scrnaseq
$ docker run scilifelablts/lts-workflows-sm-scrnaseq
$ docker run -v /path/to/workdir:/workspace -w /workspace scilifelablts/lts-workflows-
↪sm-scrnaseq -l
$ docker run -v /path/to/workdir:/workspace -w /workspace scilifelablts/lts-workflows-
↪sm-scrnaseq all
```

CHAPTER 2

Contents

2.1 Installation

Although installing *lts_workflows_sm_scrnaseq* itself should be easy enough, installing the workflow dependencies requires that you perform some manual steps. The number of steps depend on the installation route you take. For instance, the conda installation will install all python3 dependencies on the fly.

Hint: Whatever installation route you choose, it is recommended to create conda environments for python2 and python3 packages. The workflow will be executed from the python3 environment and by default is configured to load a python2 environment named *py2.7*. You can modify the python2 environment name in the configuration setting *config['conda']['python2']*.

2.1.1 Stable release from Conda (including all dependencies)

Note: Make sure your channel order complies with the instructions provided by bioconda. In short, your *~/.condarc* file should have the following channel configuration:

```
channels:
  - bioconda
  - conda-forge
```

The workflow depends on a couple of python2 packages that cannot be added as requirement to Conda. Therefore, these packages must be installed manually.

```
$ conda create -n py2.7 python=2.7 rpkmforgenes=1.0.1 rseqc=2.6.4
```

To install *lts_workflows_sm_scrnaseq*, create a new environment, activate it and install the package:

```
$ conda create -n lts-workflows-sm-scrnaseq -c scilifelab-lts lts-workflows-sm-
↪scrnaseq
```

This is the preferred method to install *lts_workflows_sm_scrnaseq*, as it will always install the most recent stable release.

2.1.2 Stable release from Conda (only workflow)

lts_workflows_sm_scrnaseq is also shipped with all the dependencies packaged in Docker/Singularity images. By running either the whole workflow or the individual jobs in containers, you can avoid having to install all the dependencies from Conda. If you'd like to run Snakemake and the workflow itself on the host system but execute the jobs in containers, then you can install a lightweight version version with:

```
$ conda create -n lts-workflows-sm-scrnaseq -c scilifelab-lts lts-workflows-sm-
↪scrnaseq-slim
```

See Usage for details on how this works.

2.1.3 From sources

The sources for lts_workflows_sm_scrnaseq can be downloaded from the Bitbucket repo.

```
$ git clone git@bitbucket.org:scilifelab-lts/lts-workflows-sm-scrnaseq.git
```

Once you have a copy of the source, you can install it with:

```
$ cd lts-workflows-sm-scrnaseq
$ python setup.py install
```

You can also install in development mode with:

```
$ python setup.py develop
```

See the section on *Development* for more information.

You can setup the python 2 packages as in the previous section, or by using the environment file *lts_workflows_sm_scrnaseq/environment-27.yaml*:

```
$ conda env create -n py2.7 -f lts_workflows_sm_scrnaseq/environment-27.yaml
```

2.1.4 Tests

If *lts_workflows_sm_scrnaseq* has been installed as a module, run

```
$ pytest -v -rs -s --pyargs lts_workflows_sm_scrnaseq
```

In order to load the pytest options provided by the module, the full path to the test suite needs to be given:

```
$ pytest -v -rs -s /path/to/lts_workflows_sm_scrnaseq/tests
```

See *Test-based development* for more information.

2.2 Usage

This section provides some simple usage examples. For more information what the workflow does, see *Single-cell RNA sequencing workflow*.

2.2.1 Running the wrapper script

lts_workflows_sm_scrnaseq comes with a wrapper script that calls snakemake and provides additional help messages. Running the wrapper without any arguments will generate a help message. If any arguments are provided they are passed on to snakemake.

```
$ lts_workflows_sm_scrnaseq
$ lts_workflows_sm_scrnaseq -l
$ lts_workflows_sm_scrnaseq all
$ lts_workflows_sm_scrnaseq --use-conda all
$ lts_workflows_sm_scrnaseq -s /path/to/Snakefile --use-conda all
```

If no Snakefile is provided, the wrapper script will automatically load the Snakefile from the package root directory (see *example_snakefile*). Note that you will have to pass a configuration file with the *–configfile* parameter.

In the case you need to add custom rules, or want to hardcode parameters such as the working directory and configuration file, you can of course copy and edit the provided Snakefile.

2.2.2 Running snakemake

You can of course bypass the provided wrapper script and run snakemake directly on your own Snakefile. If so, the intended usage is to include the main workflow file in a Snakefile. See the examples in the test directory.

```
$ snakemake -s Snakefile -d /path/to/workdir --configfile config.yaml all
```

2.2.3 Running docker/singularity containers

lts_workflows_sm_scrnaseq is also shipped with all the dependencies packaged in a Docker image. This eliminates some of the installation issues, at the cost of having to download a large image file (>5GB). In any case, the entry point of the image points to the *lts_workflows_sm_scrnaseq* wrapper script.

```
$ docker pull scilifelablts/lts-workflows-sm-scrnaseq
$ docker run scilifelablts/lts-workflows-sm-scrnaseq
$ docker run -v /path/to/workdir:/workspace -w /workspace scilifelablts/lts-workflows-
↪sm-scrnaseq -l
$ docker run -v /path/to/workdir:/workspace -w /workspace scilifelablts/lts-workflows-
↪sm-scrnaseq all
```

Docker is not allowed on many HPC systems, in which case you may be able to use Singularity instead. Docker images can, in most cases, be converted into Singularity images with:

```
$ singularity pull docker://scilifelablts/lts-workflows-sm-scrnaseq
```

This will create a file *lts-workflows-sm-scrnaseq.simg* (or similar) that is the Singularity image. This can then be started with:

```
$ singularity exec lts-workflows-sm-scrnaseq.simg
```

The method above is convenient if you want to run on a resource without internet access, where installing via Conda isn't an option. A typical use case is then to request a large node and then run the whole workflow within the Singularity container on that node. This can be inefficient, since the jobs have very different memory/cpu requirements and you're limited to however many cores are available on the node. A better, but somewhat trickier, option is to run the individual jobs in the Singularity container, but the overall workflow on the local system (see Installation for how to install this in the best way). This can be achieved with:

```
$ snakemake [your normal stuff] --use-singularity --use-conda
```

The *–use-singularity* flag tells Snakemake to execute the jobs in the Singularity container specified by *config['workflow']['singularity']*. By default this is set automatically to fit with the current release. If the dependencies for all rules could get along in the same environment, then this would be all that was needed. Unfortunately, there are some rules that require Python 2. That is what the *–use-conda* flag is for. Snakemake will then create Conda environments for those specific rules, and activate within the Singularity container. If you want this to be done for all rules, maybe to use another Singularity container, you can set *config['workflow']['use_conda_for_py3']* to *True*.

Say you are on a system without internet access and would like to submit jobs to the cluster scheduler as individual jobs. You can then combine both these methods; start the workflow in the Singularity container and then execute jobs in the same container on the compute nodes.

```
$ singularity pull docker://scilifelablts/lts-workflows-sm-scrnaseq
$ [upload lts-workflows-sm-scrnaseq.simg to cluster]
$ singularity exec lts-workflows-sm-scrnaseq.simg snakemake [your normal stuff] --use-
↪singularity --use-conda
```

Remember to set *config['workflow']['singularity']* to use the local image.

NOTE: This currently won't work for jobs that should run locally, since that would mean running a Singularity image within a Singularity image. Consider this an advanced feature that probably requires some tinkering to get running. Maybe more applicable to very large projects.

2.2.4 Example Snakefile

The provided minimum Snakefile looks as follows:

```
# -*- snakemake -*-
from lts_workflows_sm_scrnaseq import WORKFLOW

include: WORKFLOW
```

2.3 Configuration

Note: The configuration key nomenclature hasn't been settled yet

Note: The main `lts_workflows` documentation provides more information about general configuration settings.

2.3.1 Required configuration

The following options must be set in the configuration file:

```
settings:
  sampleinfo: sampleinfo.csv
  runfmt: "{SM}/{SM}_{PU}_{DT}"
  samplefmt: "{SM}/{SM}"
ngs.settings:
  db:
    ref: # Reference sequences
      - ref.fa
      - gfp.fa
      - ercc.fa
    transcripts:
      - ref-transcripts.fa
      - gfp.fa
      - ercc.fa
  annotation:
    sources:
      - ref-transcripts.gtf
      - gfp.genbank
      - ercc.gb
  # Optional; change these if read names and fastq file suffixes differ
  read1_label: "_1"
  read2_label: "_2"
  fastq_suffix: ".fastq.gz"

# list of sample identifiers corresponding to the sampleinfo 'SM'
# column
samples:
  - sample1
  - sample2
```

The configuration settings `runfmt` and `samplefmt` describe how your data is organized. They represent python miniformat strings, where the entries correspond to columns in the sampleinfo file; hence, in this case, the columns **SM**, **PU** and **DT** must be present in the sampleinfo file.

Note: Since the `runfmt` and `samplefmt` can represent any format you wish, in principle, you could use any label formatting names. This is true except for **SM**, which represents the sample name and **must** be present in the sampleinfo file. The two-letter sample labels above are convienient representations of metadata and correspond to samtools read group record types.

Example sampleinfo.csv

```
SM,PU,DT,fastq
s1,AAABBB11XX,010101,s1_AAABBB11XX_010101_1.fastq.gz
s1,AAABBB11XX,010101,s1_AAABBB11XX_010101_2.fastq.gz
s1,AAABBB22XX,020202,s1_AAABBB22XX_020202_1.fastq.gz
s1,AAABBB22XX,020202,s1_AAABBB22XX_020202_2.fastq.gz
s2,AAABBB11XX,010101,s2_AAABBB11XX_010101_1.fastq.gz
s2,AAABBB11XX,010101,s2_AAABBB11XX_010101_2.fastq.gz
```

The example sampleinfo file would work with the required settings above. The following `runfmt` and `samplefmt` would be generated for sample *s2*, read 1:

```
runfmt = s2/s2_AAABBB11XX_010101
samplefmt = s2/s2
```

2.3.2 Workflow specific configuration

In addition to the required configuration, there are some configuration settings that affect the workflow itself. These settings are accessed and set via `config['workflow']`.

use_multimapped (boolean) Use multimapped reads for quantification. Default *false*.

quantification (list) List quantification methods to use. Available options are *rsem* and *rpkmforgenes*.

Example workflow configuration section

```
workflow:
  use_multimapped: false
  quantification:
    - rsem
    - rpkmforgenes
```

2.3.3 Application level configuration

Note: Unfortunately, there is no straightforward way to automatically list the available application configuration options. You therefore have look in the rule files themselves for available options. In most cases, the default settings should work fine.

Note: Rules live in separate files whose names consist of the application name followed by the rule name. Rules are located in package subdirectory *rules*, in which each application lives in a separate directory.

Tip: There is a `option` configuration key for each rule. Most often, this is the setting one wants to modify.

Individual applications (e.g. star) are located at the top level, with sublevels corresponding to specific application rules. For instance, the following configuration would affect settings in `star` and `rsem`:

```
star:
  star_index:
    # The test genome is small; 2000000 bases. --genomeSAindexNbases
    # needs to be adjusted to (min(14, log2(GenomeLength)/2 - 1))
    options: --genomeSAindexNbases 10

rsem:
  index: ../ref/rsem_index
```

2.3.4 Additional advice

There are a couple of helper rules for generating spikein input files and the transcript annotation file.

`dbutils_make_transcript_annot_gtf` For QC statistics calculated by RSEQC, the gtf annotation file should reflect the content of the alignment index. You can automatically create the file name defined in `['ngs.settings']['annotation']['transcript_annot_gtf']` from the list of files defined in `['ngs.settings']['annotation']['sources']` via the rule `dbutils_make_transcript_annot_gtf`. gtf and genbank input format is accepted.

ercc_create_ref The ERCC RNA Spike-In Mix is commonly used as spike-in. The rule `ercc_create_ref` automates download of the sequences in fasta and genbank formats.

2.4 Troubleshooting

2.4.1 The workflow cannot find RSeQC or rpkmforgenes

Make sure you have created a conda environment for python2 packages whose name matches that of the configuration setting *config['conda']['python2']* (default is `py2.7`).

2.4.2 STAR align cannot find the input files even though they are present

The input fastq file names depend on the configuration setting *config['settings']['runfmt']* as well as *config['ngs.settings']['read1_label']* (*read2_label* for read 2) and *config['ngs.settings']['fastq_suffix']*. Make sure that read labels and fastq suffix are correctly configured.

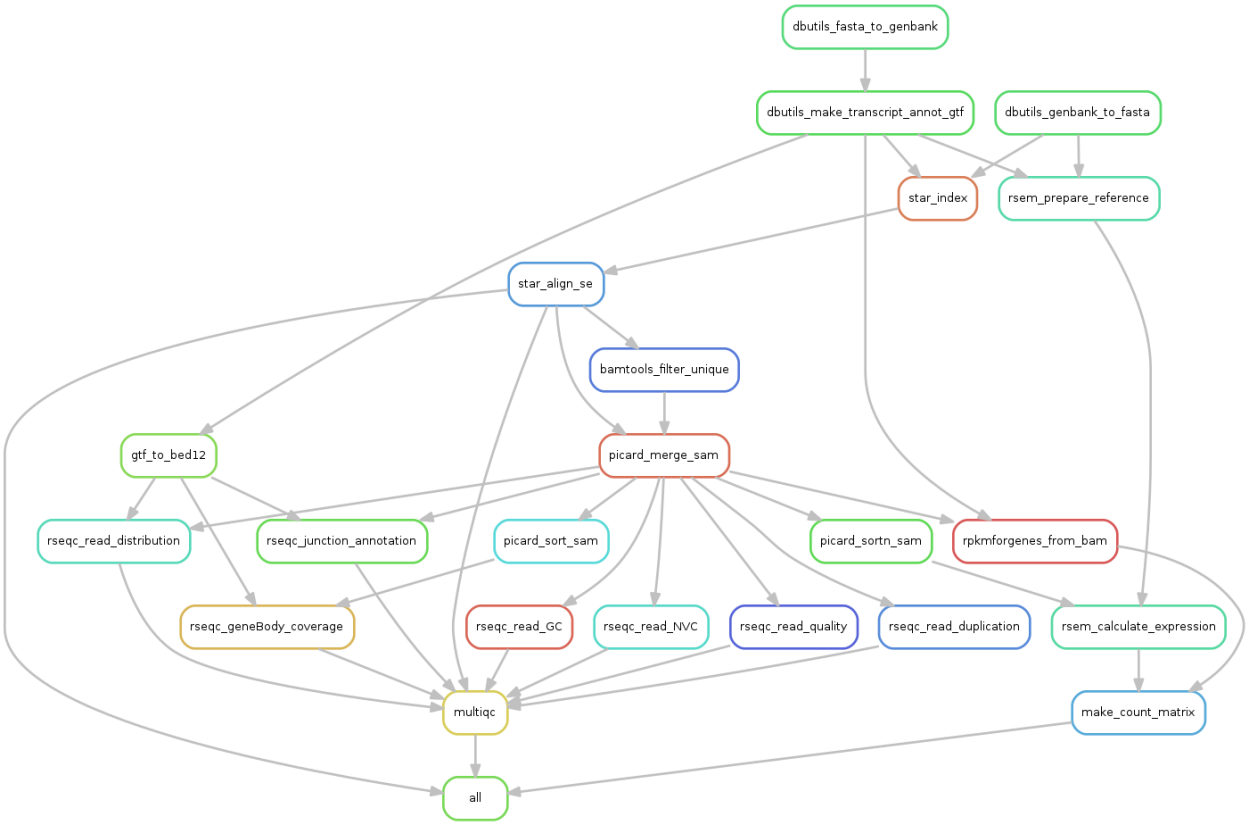
2.5 Single-cell RNA sequencing workflow

This workflow does the following:

1. aligns reads with STAR to a genome or transcriptome reference
2. gene/transcript quantification with RSEM and/or rpkmforgenes
3. basic qc with RSeQC and MultiQC
4. filters out failed cells based on the qc stats and generates a report

2.5.1 Workflow

The figure below illustrates the workflow included in the test directory. Here, use has been made of the additional rule mentioned in *Additional advice* to generate the transcript annotation gtf.



2.6 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

2.6.1 Types of Contributions

Report Bugs

Report bugs at https://bitbucket.org/scilifelab-lts/lts-workflows-sm-scrnaseq/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the Bitbucket issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

Implement Features

Look through the Bitbucket issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

lts_workflows_sm_scrnaseq could always use more documentation, whether as part of the official lts_workflows_sm_scrnaseq docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://bitbucket.org/scilifelab-lts/lts-workflows-sm-scrnaseq/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

2.6.2 Get Started!

Ready to contribute? Here's how to set up *lts_workflows_sm_scrnaseq* for local development.

1. Fork the *lts_workflows_sm_scrnaseq* repo on Bitbucket.
2. Clone your fork locally:

```
.. code-block:: console

   $ git clone git@bitbucket.org:your_name_here/lts-workflows-sm-scrnaseq.git
```

3. Follow the instructions in *Development* to install a conda development environment and the package.
4. Create a branch for local development:

```
.. code-block:: console

   $ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests:

```
$ make lint
$ pytest -v -s -rs
```

6. Commit your changes and push your branch to Bitbucket:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the Bitbucket website.

2.6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests (optional).
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature HISTORY.rst.
3. Check https://bitbucket.org/scilifelab-lts/lts-workflows-sm-scrnaseq/addon/pipelines/home and make sure that the CI tests pass.

2.7 Development

If you want to contribute to the development of *lts_workflows_sm_scrnaseq*, you need to setup a development enviromnent and follow some basic guidelines. In addition, there are workflow tests that provide convenient help for test-based development. This section details some of this information.

2.7.1 Development environment

Setting up a conda development environment

Start by setting up the main conda development environment for python3 packages. This is where the source code will live.

```
$ conda env create -n lts-workflows-sm-scrnaseq-devel python=3.6
```

For the workflow tests, you also need a conda environment for python2 packages:

```
$ conda env create -n py2.7 python=2.7
```

Hint: By default, the workflow tests are configured to use a python 2 environment named *py2.7*. Should you name it otherwise, you can tell pytest with the flag `--python2-conda`:

```
pytest -v -s --python2-conda mypython2env lts_workflows_sm_scrnaseq/tests/test_
↪workflows_sm_scrnaseq.py
```

Finally, activate the development environment:

```
$ source activate lts-workflows-sm-scrnaseq-devel
```

Hint: Instead of having to remember when to activate a specific environment, maybe also setting environment variables in the process, we recommend using direnv. With direnv, you can add a hidden file *.envrc* to the source root directory with the following content:

```
source activate lts-workflows-sm-scrnaseq-devel
export DOCKER_REPO=scilifelablts
```

Now, every time you cd into the source directory, the correct environment will be activated. Likewise, the environment will be deactivated once you cd out of the source directory.

Checkout the source repository and install

Once the conda environments are setup, checkout the source repository. If you haven't done so already, fork the *lts_workflows_sm_scrnaseq* bitbucket repository.

```
$ git clone git@bitbucket.org:your_name_here/lts-workflows-sm-scrnaseq.git
```

Change directory to the source code directory and install in **development mode**:

```
$ cd lts-workflows-sm-scrnaseq
$ python setup.py develop
```

Install development requirements

Install the package requirements for development with pip:

```
$ pip install -r requirements_dev.txt
```

Note: In principle, conda should work as follows:

```
$ conda install --file requirements_dev.txt
```

However, for some reason the `docker-py` module is named docker on PyPI and docker-py on anaconda, causing the install to fail.

If everything works ok, you should now have a operating development environment! Now you are ready to create and test your new features. Follow the guidelines in *Get Started!* to submit a pull request.

2.7.2 Test-based development

lts_workflows_sm_scrnaseq includes tests to ensure code and workflow integrity. Some tests are run on the codebase itself (located in the *tests* directory), whereas others execute pipeline runs (located in the *lts_workflows_sm_scrnaseq/tests* directory). The workflow tests are shipped with the package. The development environment requirements include the module `pytest-ngsfixtures`, a pytest which provides pytest fixtures in the shape of next generation sequencing test data sets.

The tests are located in the *lts_workflows_sm_scrnaseq/tests* directory. You can execute all the tests by executing

```
$ pytest -v -s -rs lts_workflows_sm_scrnaseq/tests
```

Note: Since the tests reside in the module directory, they are actually installed as a part of the module upon installation. This means you (theoretically) can run the tests like this:

```
$ pytest -v -s -rs --pyargs lts_workflows_sm_scrnaseq
```

Workflow tests

When developing, you are most likely only interested in runnig the workflow tests in *lts_workflows_sm_scrnaseq/tests/test_lts_workflows_sm_scrnaseq.py*. Running

```
$ pytest -v -s -rs lts_workflows_sm_scrnaseq/tests/test_lts_workflows_sm_scrnaseq.py
```

will run a listing of the workflow as well as an entire workflow run on two samples.

You can control some aspects of the tests by applying module-specific pytest options. The conftest plugin `lts_workflows_sm_scrnaseq.tests.conftest` is a pytest local per-directory plugin that among other things adds command line options to pytest.

Note: For some reason, these options disappear when running the test from the source root directory, or with the *–pyargs* option.

For instance, the option *–ngs-test-command* controls whether workflow list or run is executed. The following command will only run the workflow.

```
$ cd lts_workflows_sm_scrnaseq/tests
$ pytest -v -s -rs test_lts_workflows_sm_scrnaseq.py --ngs-test-command run
```

A full list of test-specific options is given here:

```
 single cell rna sequencing options:
--no-slow             don't run slow tests
-H, --hide-workflow-output
                      hide workflow output
-T THREADS, --threads=THREADS
                      number of threads to use
--enable-test-conda   enable test conda setup; automatically install all
                      dependencies in semi-persistent test conda
                      environments
--conda-install-dir=CONDA_INSTALL_DIR
                      set conda install dir
--conda-update        update local conda installation
-2 PYTHON2_CONDA, --python2-conda=PYTHON2_CONDA
                      name of python2 conda environment [default: py2.7]
-C, --use-conda       pass --use-conda flag to snakemake workflows; will
                      install conda environments on a rule by rule basis
--ngs-test-unit=unit [unit ...]
                      test unit - test source code locally or distributed in
                      docker container (choices: local,docker)
--smode={se} [{se} ...], --sequencing-modes={se} [{se} ...]
                      sequencing modes (paired end or single end) to run
--ngs-test-command=command [command ...]
                      test command - run or list
```

Rerunning workflow tests in pytest tmp directory

By default, pytest outputs test results in numbered subdirectories of directory */tmp/pytest-of-user/*. In order to facilitate test-based development, for each workflow test, `pytest-ngsfixtures` outputs a file *command.sh* in the resulting test directory. For instance, if the workflow test has been run, this file should exist in a subdirectory *run-se-local* in one of the numbered subdirectories of */tmp/pytest-of-user*. *command.sh* is an executable that records the environment and command used to run the test. Therefore, if a workflow run has failed, the test can be rerun as follows:

```
$ cd /tmp/pytest-of-user/pytest-#/run-se-local
$ ./command.sh
```

Consequently, this is a convenient way of setting up a current test data environment for the workflow. Fix the code and rerun *command.sh* until the test passes.

Hint: By default, *command.sh* runs the *all* target. Edit the file to your liking to target other outputs of interest when developing a new feature.

Testing the docker images

Apart from providing source code and conda packages, `lts_workflows_sm_scrnaseq` also provides Docker images at https://hub.docker.com/r/scilifelablts/lts-workflows-sm-scrnaseq/.

Important: The docker tests use docker swarm mode to maintain and manage running containers. In order to do test development on the Dockerfile, you first need to initialize docker swarm; see docker create swarm for more information.

When developing the Dockerfile and the resulting images, you can use the make commands. For instance, to build and test changes to the main Dockerfile, run (assuming you're standing in the source root directory):

```
$ make docker
$ pytest -v -s -rs lts_workflows_sm_scrnaseq/tests/test_lts_workflows_sm_scrnaseq.py -
↪-ngs-test-unit docker
```

The option `ngs-test-unit` selects what test unit to run, either *local* (i.e. local source code) or *docker* which tests the docker container. The test will fire up a docker stack and provide a docker service named *lts_scrnaseq_snakemake*, which you can see by running the command

```
$ docker service ls
```

Note: Developing with docker swarm can at first be confusing since on some occasions, updates to the local repository are **not** present in an updated image. This has to do with how docker handles version tags and repositories. It may help to restart services, but it may also be necessary to use a local registry (usually *localhost:5000*) and push updated images to this registry. See the docker documentation for more information.

2.8 Credits

2.8.1 Development Lead

- Per Unneberg <per.unneberg@scilifelab.se>
- Rasmus Ågren <rasmus.agren@scilifelab.se>
- Leif Väremo <leif.varemo@scilifelab.se>
- Åsa Björklund <asa.bjorklund@scilifelab.se>

2.8.2 Contributors

None yet. Why not be the first?

2.9 History

2.9.1 0.3.6 (2018-10-05)

- Bumped Snakemake version to deal with permission problems when using the *script* directive.

2.9.2 0.3.5 (2018-05-29)

- Added the option to execute jobs in a singularity container
- Added lts-workflows-sm-scrnaseq-slim Conda package, which is only the workflow without the tools. To be used with "–use-singularity"
- Modified Docker images to install to "base" environment rather than to "snakemake"

2.9.3 0.3.4 (2018-05-25)

- Added the option to group sets of cells and run as group jobs
- Changed to use scater v1.6 and SingleCellExperiment
- Changed to use "runtime" as a rule resource rather than a parameter

2.9.4 0.3.3 (2018-05-07)

- Rewrote QC report from scratch
- Added filtering of contaminating cells based on marker gene expression
- Changed QC config structure
- No longer built for Python 3.5 (dumping settings to yaml relies on that dicts are ordered, which was implemented in 3.6)

2.9.5 0.3.2 (2018-04-23)

- Rewrote QC report to fit better as part of a workflow
- Added biotypes to the QC and filtering steps
- Removed unnecessary logs to reduce the number of files created
- Explicitly include all files generated by the workflow rules
- Flag more files as temporary to reduce space usage
- General clean up of the code

2.9.6 0.3.1 (2018-04-17)

- Changed to use R-markdown 1.8 due to error in the conda-forge recipe for 1.5.

2.9.7 0.3.0 (2018-04-09)

- Added QC and filtering
- Changed CI and testing Docker images

2.9.8 0.2.0 (2017-01-30)

- Simplify test setup
- Make pytest-ngsfixtures optional
- Use picard instead of samtools rules for sorting
- Update rseqc rule for empty fastq input (#37)
- Add multiqc rule (#33)
- Add rule for rpkm/count matrix (#31)
- Add gene entry to gtf file (#28)

2.9.9 0.1.0 (2017-02-12)

- First release on conda.

2.10 lts_workflows_sm_scrnaseq

2.10.1 lts_workflows_sm_scrnaseq package

Subpackages

lts_workflows_sm_scrnaseq.core package

Submodules

lts_workflows_sm_scrnaseq.core.utils module

lts_workflows_sm_scrnaseq.core.utils.**get_samples**(*config*, *logger*)

lts_workflows_sm_scrnaseq.core.utils.**python2_path**(*config*, *logger*)
Add python 2 path if possible

lts_workflows_sm_scrnaseq.core.utils.**to_min**(*timestr*)

lts_workflows_sm_scrnaseq.core.wrappers module

lts_workflows_sm_scrnaseq.core.wrappers.**lts_workflows_sm_scrnaseq_wrapper**()
Wrapper for running lts_workflows_sm_scrnaseq workflow. Any argument will be passed to snakemake. Consequently, this means you *must* supply a workflow target to run the workflow. By default, the wrapper will use a generic Snakefile shipped with the package. Note that in this case you must supply a configuration file via the –configfile option.

Examples

```
$ lts_workflows_sm_scrnaseq -l
$ lts_workflows_sm_scrnaseq all --configfile config.yaml -d /path/to/workdir
```

If the docker image is used to run the workflow, this wrapper serves as the entry point. The image uses gosu to set the user id of the main process, which defaults to user id 9001. In order to run as the local user, the environment variable LOCAL_USER_ID must be passed to the docker run process (recommended).

Examples

```
$ docker run scilifelablts/lts-workflows-sm-scrnaseq
$ docker run -v /path/to/workdir:/workspace -w /workspace scilifelablts/lts-
↪workflows-sm-scrnaseq all --configfile config.yaml
$ docker run -e LOCAL_USER_ID=1000 -v /path/to/workdir:/workspace -w /workspace␣
↪scilifelablts/lts-workflows-sm-scrnaseq all --configfile config.yaml
$ docker run -e LOCAL_USER_ID=1000 -v /path/to/workdir:/workspace -w /workspace --
↪entrypoint "/bin/bash" scilifelablts/lts-workflows-sm-scrnaseq
```

All commands are handled by the lts_workflows_sm_scrnaseq wrapper, but you can also explicitly call snakemake:

```
$ docker run -v /path/to/workdir:/workspace -w /workspace scilifelablts/lts-
↪workflows-sm-scrnaseq snakemake all --configfile config.yaml
```

The wrapper runs a package Snakefile with the following minimum content:

```
from lts_workflows_sm_scrnaseq import WORKFLOW
include: WORKFLOW
```

If need be, extend this file with custom rules and directives and run it with the wrapper or as usual with regular Snakemake.

Module contents

Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

Index