cgatcore Documentation

Release 1.0

CGAT Developers

Jan 10, 2019

Getting started

cgat-showcase contains a basic RNA-seq differential expression pipeline to illustrate how to build workflows using the **cgat-core** workflow management system. cgatcore allows you to quickly build both simple and complex pipelines for reproducible data analysis within a python script.

cgat-core is a set of libraries and helper functions that enable researchers to quickly design and build computational workflows for the analysis of large-scale data-analysis. Access to code is through github and documentation can be accessed here.

Used in combination with cgat-apps, we have deomonstrated the functionality of our flexible implementation using a set of well documented, easy to install and easy to use workflows, which can be found in our cgat-flow repository.

However, in this repository we have developed a toy example showing the functionality of our code. For further information on the advanced functionality of our code please refer to the cgat-flow docuemntation.

cgat-core is a user-friendly and powerful open-source workflow management system and has been continually developed and used for implemention Next Generation Sequencing (NGS) pipelines for over 10 years.

CHAPTER 1

Citation

To be added. . . .

CHAPTER 2

Support

- Please refer to our *FAQs* section
- In case of questions, please add these to stack overflow
- For bugs and issues, please raise an issue on github

2.1 Installation

The following sections describe how to install the cgatshowcase framework. Since there are very minimal dependancies the installation of the code should not take too long. The main dependancies are cgat-core, cgat-apps, kallisto, deseq2, biomart, tximport and genomeinfodb.

2.1.1 Installation using conda

The most convinient way to use our code is to install the software using conda. First you will need to install conda using miniconda or anaconda and following the instructions here. (we recomend miniconda rather than anaconda)

Once you have conda installed then to install cgat-showcase please do the following:

```
conda install -c cgat cgatshowcase
```

This should install all of the dependancies and you should be read to proceded to the Writing workflows and Tutorial section. However, conda is currently having issues with speed of installation. This is related to the solver and is something that is known and the conda developers are workin on new fixes for this. Please see conda issue for more information.

2.1.2 Pip installation

Alternatively, cgatshowcase can also be installed using pypi:

```
pip install cgatshowcase
```

However, the dependancies will have to be installed seperately.

2.1.3 Manual installation

To obtain the latest code, check it out from the public git repository and activate it:

```
git clone https://github.com/cgat-developers/cgat-core.git
cd cgat-core
python setup.py develop
```

Once checked-out, you can get the latest changes via pulling:

```
git pull
```

2.1.4 Installing additonal software

When building your own workflows we recomend using conda to install software into your environment where possible.

This can easily be performed by:

```
conda search <package>
conda install <package>
```

2.2 Running a pipeline - Tutorial

Before beginning this tutorial plase make sure you have cgat-showcase installed correctly, please see here (see *Installation*) for instructions.

As a tutorial example we have written a showcase example of how you can build robust and scalable pipelines written in python. For this toy example we have written a workflow that performs pseudocounting using kallisto and differential expression analysis using deseq. Two reports are then generated, one using MultiQC and another written in Rmarkdown. However, because of the flexible nature of our workflow management system, any reporting strategy can be implimented.

2.2.1 Tutorial

1. First download the tutorial data:

```
mkdir showcase
cd showcase
wget https://www.cgat.org/downloads/public/showcase/showcase_test_data.tar.gz
tar -zxvf showcase_test_data.tar.gz
```

2. Next we will generate a configuration yml file so the pipeline output can be modified:

```
cd showcase_test_data
cgatshowcase transdiffexprs config
```

or you can alternatively call the workflow file directly:

```
python /path/to/file/pipeline_transdiffexprs.py config
```

This will generate a **pipeline.yml** file containing the configuration parameters than can be used to modify the output of the pipleine. However, for this tutorial you do not need to modify the parameters to run the pipeline. In the modify_config section below I have detailed how you can modify the config file to change the output of the pipeline.

3. Next we will run the pipleine:

```
cgatshowcase transdiffexprs make full -v5 --no-cluster
```

This `--no-cluster` will run the pipeline locally if you do not have access to a cluster. Alternatively if you have a cluster remove the `--no-cluster` option and the pipleine will distribute your jobs accross the cluster.

Note: There are many commandline options available to run the pipeline. To see available options please run `cgatshowcase --help`.

4. Generate a report

The final step is to generate a report to display the output of the pipeline. We have a preference for using MultiQC for generate bioinformatics tools (such as mappers and pseudoaligners) and Rmarkdown for generating custom reports. In order to generate these run the command:

```
cgatshowcase transdiffexprs make build_report -v 5 --no-cluster
```

This will generate a MultiQC report in the folder *MultiQC_report.dir/* and an Rmarkdown report in *R_report.dir/*.

This completes the tutorial for running the transdiffexprs pipeline for cgat-showcase, hope you find it as useful as we do for writing workflows within python.

2.2.2 Modify the configuration file

Having generated a default pipeline.yml file, you may require the output of the pipeline to be modified. Foe example, if you require a different genome or a different fdr value for significance. These can be changed modifying the yml file and adding user specific infomation.

For tutorial purposes we will modify the fdr value to specify a more stringent cuttoff value.

The original pipeline.yml file for deseq2 parameters is as follows:

```
#################################################################
# Deseq2 options
#################################################################

deseq2:
    # fdr to accept for deseq2
    fdr: 0.05

    # model to pass as DESeq2 design
    model: ~group

    # contrast to return during post-hoc pairwise tests
       contrast: group

       # reference level for contrast , e.g WT
```

(continues on next page)

(continued from previous page)

```
       control: Brain1
       # test for significance for deseq1 - wald or lrt
       detest: wald
```

In order to change the fdr settings you will need to delete the directory *DEresults.dir*, since ruffus is checking for the presence of the output of the run_deseq2 function. Removing this directory will let ruffus know that this function needs to be re-ran.:

```
rm -rf DEresults.dir/
```

You can check to see the pipeline graph and see that the task run_deseq is now waiting to be ran by running:

```
cgatshowcase plot full -v5
```

Now the pipeline.yml file can be modified to make the fdr more conservative, as follows:

```
################################################################
# Deseq2 options
################################################################

deseq2:
    # fdr to accept for deseq2
    fdr: 0.05

    # model to pass as DESeq2 design
    model: ~group

    # contrast to return during post-hoc pairwise tests
       contrast: group

       # reference level for contrast , e.g WT
       control: Brain1
       # test for significance for deseq1 - wald or lrt
       detest: wald
```

Then the pipeline can be re-ran

```
cgatshowcase transdiffexprs make full -v 5 --no-cluster
```

All tasks downstream of the run_deseq will be re-ran.

2.3 Writing a pipeline - Tutorial

Before attempting this tutorial please make sure you have cgat-showcase installed correctly. please see here (see *Installation*) for instructions.

This tutorial is intended as a walkthrough to show you the processes inviolved in writing the cgat-showcase transdiffexprs pipeline. It wont go through all of the functions in the pipeline but it should give you an idea of how to construct a pipeline using our cgat-core python library. Before attempting this tutorial, it may be best to visit cgat-core documentation for an idea of how to construct a very simple pipeline.

2.3.1 Tutorial

1. The problem and the solution

The first step in any pipeline is to think about the problem that you are trying solve. For example:

The problem: I need to perform differential expression analysis quickly and consistently across all my samples. I also need the solution to scale and be ran on my mac or districuted across a cluster. **The solution:** Build a cgat-core workflow that wraps pseduocounting tools (kallisto) and differential expression analysis tools (deseq2).

Then the next step is to think about the steps needed to reach the solution.

For example:

1. Build a reference transcriptome from a gtf
2. Build a kallisto index of that transcriptome
3. Perform pseudocounting of the fasta sequencing data
4. Run differential expression analysis using deseq2
5. Generate a report of the pseudocounting quality
6. Generate a report displaying the differential expression output

Then the next step is to begin to code the pipeline. There are certain core modules that will help with the building of our workflows and they are described below.

2. Setting up the pipeline

First navigate to a directory where you would like to start building the pipeline then create a directory called cgat-showcase:

```
mkdir cgat-showcase && mkdir cgat-showcase/cgatshowcase && touch cgat-showcase/
↪cgatshowcase/pipeline_transcriptdiffexprs.py && mkdir cgat-showcase/cgatshowcase/
↪pipeline_transcriptdiffexprs && touch cgat-showcase/cgatshowcase/ModuleTrans.py
```

This will create a directory with the following input:

```
|-- cgat-showcase
   |-- cgatshowcase
      -- pipeline_transdiffexprs.py
      -- ModuleTrans.py
      |-- pipeline_transdiffexprs
```

The layout has the following components:

pipeline_transdiffexprs.py This is the file that will contain all of the ruffus workflows, the file needs the format pipeline_<name>.py

pipeline_transdiffexprs/ Directory containing the configuration yml file. The directory needs to be named the same as the pipeline_<name>.py file

ModuleTrans.py This file will contain functions that will be imported into the main ruffus workflow file, pipeline_transdiffexprs.py

3. Creating basic utility functions

Open the cgat-shocase/cgatshowcase/pipeline_transdiffexprs.py file and start populating code with basic utility functions to help with building workflows.

The code begins with a doc string detailing the pipeline functionality.You should use this section to document your pipeline.:

```
'''This pipeline is a test and this is where the documentation goes '''
```

Config parser: This code will help with parsing the pipeline.yml file:

```
# load options from the config file
P.get_parameters(
    ["%s/pipeline.yml" % os.path.splitext(__file__)[0],
     "../pipeline.yml",
     "pipeline.yml"])

PARAMS = P.PARAMS
```

Commandline parser: This bit of code allows pipeline to parse arguments and run the pipleine using the *cgatshowcase* command:

```
def main(argv=None):
    if argv is None:
        argv = sys.argv
    P.main(argv)


if __name__ == "__main__":
    sys.exit(P.main(sys.argv))
```

4. Create the ruffus functions

The first ruffus function will build a reference transcriptome so kallisto can build an index.

For this to be acomplished the following code is written:

```
@mkdir('geneset.dir')
@transform(PARAMS['geneset'],
        regex("(\S+).gtf.gz"),
        r"geneset.dir/\1.fa")
def buildReferenceTranscriptome(infile, outfile):
    '''
    Builds a reference transcriptome from the provided GTF geneset - generates
    a fasta file containing the sequence of each feature labelled as
    "exon" in the GTF.
    --fold-at specifies the line length in the output fasta file
    Parameters
    ----------
    infile: str
        path to the GTF file containing transcript and gene level annotations
    genome_dir: str
        :term: `PARAMS` the directory of the reference genome
    genome: str
        :term: `PARAMS` the filename of the reference genome (without .fa)
    outfile: str
```

(continues on next page)

(continued from previous page)

```
        path to output file
    '''

    genome_file = os.path.abspath(
        os.path.join(PARAMS["genome_dir"], PARAMS["genome"] + ".fa"))

    statement = '''
    zcat < %(infile)s |
    awk '$3=="exon"'|
    cgat gff2fasta
    --is-gtf --genome-file=%(genome_file)s --fold-at=60 -v 0
    --log=%(outfile)s.log > %(outfile)s;
    samtools faidx %(outfile)s
    '''

    P.run(statement)
```

Subsequent functions can then be written to form the workflow of the pipeline. More infomation on how to build ruffus pipelines can be found on the ruffus documentation and cgat-core documentation. Please see the source code in pipeline_tranmsdiffexprs.py for more infomation on the specific ruffus functions that have been written for the current pipeline.

5. Creating a pipeline.yml

There are different ways to pass configuration values to modify the output of a pipeline. For example, the threshold of the padj filtering in deseq2 can be modified. In order to achieve this we modify a configuration file, pipeline.yml. This needs to be created in the pipeline_transdiffexprs/ directory.:

```
touch &&  cgat-showcase/cgatshowcase/pipeline_transcriptdiffexprs/pipeline.yml
```

The configuration file then needs to be written so that the yml parser can pick up different options. For example,:

```
kallisto:
 kmer: 31
```

The kmer will then have the value of 31 and can be passed into then pipeline. This is accessed within the pipeline as PARAMS['kallisto_kmer'] and will have the value 31.

6. Generating a report

In order to present our results in a visually appealing manner, we use multiQC reports to display generic sequencing metrics and rmarkdown (or ipython if you prefer) for generating bespoke reports.

In order to write a report you will need to generate an rmarkdown document folder as follows:

```
mkdir cgat-showcase/cgatshowcase/pipeline_docs/pipeline_transdiffexprs/R_report
```

Rmarkdown reports are best constructed using rstudio as they can be written dynamically, then best tested within the build functionality within rstudio.

In order to build a Rmarkdown website we usually follow the Rmarkdown tutorial

The basic requirements for building a website are:

```
index.Rmd
_site.yml
R_report.Rproj
```

Once the report has been developed in rstudio and tested then the pipeline requires a function to copy and then render the report:

```
@follows(mkdir("R_report.dir"))
@follows(run_deseq2)
def run_rmarkdown_report():
    '''This will generate a rmarkdown report '''

    report_path = os.path.abspath(os.path.join(os.path.dirname(__file__),
                                               'pipeline_docs',
                                               'pipeline_transdiffexprs',
                                               'R_report.dir'))

    statement = '''cp %(report_path)s/* R_report.dir ; cd R_report.dir ; R -e
↪"rmarkdown::render_site(encoding = 'UTF-8')"'''
    P.run(statement)
```

This function copies the R_report.dir from the main repository and then calles rmarkdown to render the website according to the _site.yml configuration file.

2.4 Developers

The following individuals are the main developers of the cgat-developers

Andreas Heger

Adam Cribbs

Sebastian Luna Valero

Hania Pavlou

David Sims

Charlotte George

Tom Smith

Ian Sudbery

Jakub Scaber

Mike Morgan

Katy Brown

Nick Ilott

Jethro Johnson

Katherine Fawcett

Steven Sansom

Antonio Berlanga

cgat-showcase was written and maintained by Adam Cribbs

2.5 FAQs

As our workflow develops we will add frequently asked questions here.

In the meantime please add issues to the github page

2.6 Licence

CGAT-showcase is an open-source project and we have made the cgat-developers repositor available under the open source permissive free MIT software licence, allowing free and full use of the code for both commercial and non-commercial purposes. A copy of the licence is shown below:

2.6.1 MIT License