grape.recipe.pipeline Documentation

Release 1.1.13

Maik Röder

December 24, 2012

CONTENTS

Grape (Grape RNA-Seq Analysis Pipeline Environment) is a pipeline for processing and analyzing RNA-Seq data developed at the Bioinformatics and Genomics unit of the Centre for Genomic Regulation (CRG) in Barcelona.

The `grape.buildout` package makes use of the `grape.recipe.pipeline` recipe to configure Grape pipelines. You get preconfigured start and execute scripts, and don't have to worry about command line options any more. This makes configuring multiple Grape pipelines more convenient.

To learn more about Grape, and to download and install it, go to the Bioinformatics and Genomics website at:

Grape Homepage

Note: The `grape.recipe.pipeline` package is a Buildout recipe used by `grape.buildout`, and is not a standalone Python package. It is only going to be useful as installed by the `grape.buildout` package.

CHAPTER

ONE

MOTIVATION

Here at the CRG, we configure all our RNASeq pipeline runs in a central place before running them. Once all the accessions and pipeline profiles have been defined and the buildout parts have been created, we start and execute them on an SGE cluster.

When we receive `FASTQ` or `BAM` files for a project, we typically have to:

1. Define the accessions and profiles:

   ```
   grape.buildout/accessions/MyProject/db.cfg
   grape.buildout/profiles/MyProject/db.cfg
   ```

2. Create a pipeline project folder:

   ```
   grape.buildout/pipelines/MyProject
   ```

3. Configure the buildout:

   ```
   grape.buildout/pipelines/MyProject/buildout.cfg
   ```

4. Run the buildout in:

   ```
   grape.buildout/pipelines/MyProject
   ```

5. Run the pipelines in:

   ```
   grape.buildout/pipelines/MyProject/parts/*/
   ```

The `grape.recipe.pipeline` recipe plays a major role in step number 4. The buildout uses the recipe to produce the individual pipelines and preconfigure the start and execute scripts with all the necessary command line options.

CHAPTER

TWO

ACCESSION CONFIGURATION PARAMETERS

The accession parameters are mostly derived from UCSC's ENCODE controlled vocabulary.

The following parameters are used when configuring accessions:

`file_location`	The full path to the file If there is more than one file, put one file per line

For each line in `file_location`, a corresponding line needs to be specified for the following parameters:

`mate_id`	Using this label, files can be marked as belonging to one read
`pair_id`	Using this label, files can be associated with pairs
`label`	Using this label, files can be associated with the accession itself

The pairing information is required:

`paired`	Set this to `0` for unpaired reads, an `1` for paired.

Instead of risking to wrongly deduce the file type from the file extensions contained in the `file_location` parameter, it should be given explicitly here.

`type`	Set the file type. This can be set to `fastq` or `bam`

The qualities and the read type are important, because depending on them Grape may produce different results:

`qualities`	The base quality can be se to `phred` and `solexa`, or if you don't know the quality, you can set it to `ignore`
`readType`	Paired/Single reads lengths: Specific information about cDNA sequence reads including length, directionality and single versus paired read. See UCSC's ENCODE controlled vocabulary for readType.

The `replicate` and `species` parameters don't change anything in the behaviour of Grape, but are considered essential meta data.

`replicate`	The biological replicate of a particular experiment, if the experiment is a bioreplicate
`species`	The species, for example `Homo sapiens`, `Mus musculus`

The following parameters have been used in the ENCODE project. It makes sense to fill in the `cell` parameters and the `rnaExtract` as well. For most projects, the `localization` is probably going to be 'cell'.

`cell`	Cell, tissue or DNA sample: Cell line or tissue used as the source of experimental material. See UCSC's ENCODE controlled vocabulary for cell.
`rnaExtract`	RNA Extract: Fraction of total cellular RNA selected for by an experiment. This includes size fractionation (long versus short) and feature fractionation (PolyA-, PolyA+, rRNA-). See UCSC's ENCODE controlled vocabulary for rnaExtract.
`localization`	Cellular compartment: The cellular compartment from which RNA is extracted. Primarily used by the Transcriptome Project. See UCSC's ENCODE controlled vocabulary for localization.

CHAPTER

THREE

EXAMPLE ACCESSION CONFIGURATION

Here's a complete example of how the pipelines are configured, taken from the Test project in grape.buildout.

First we define an accession in:

```
accession/Test/db.cfg
```

This is the content of the db.cfg file:

```
[TestRun]
species = Homo sapiens
readType = 2x76
cell=NHEK
rnaExtract=LONGPOLYA
localization=CELL
replicate=1
qualities=solexa
type=fastq
file_location = ${buildout:directory}/src/testdata/testA.r2.fastq.gz
                ${buildout:directory}/src/testdata/testA.r1.fastq.gz
                ${buildout:directory}/src/testdata/testB.r2.fastq.gz
                ${buildout:directory}/src/testdata/testB.r1.fastq.gz
mate_id = testA.2
          testA.1
          testB.2
          testB.1
pair_id = testA
          testA
          testB
          testB
label = Test
        Test
        Test
        Test
type = fastq
paired = 1
```

CHAPTER

FOUR

PROFILES CONFIGURATION PARAMETERS

The following parameters are configured in the `profiles` folder, and specify the general parameters of the Grape pipeline.

The project id should be as short as possible.

`PROJECTID`	Name of the project

There are two predefined pipeline templates, one for fastq files as input and one for bam files.

`TEMPLATE`	Path to the template defining the pipeline steps For `FASTQ` files as input: ${buildout:directory}/src/pipeline/template3.0.txt For `BAM` files as input: ${buildout:directory}/src/pipeline/template.bam.txt

There are some technical settings that need to be made so that the results are written to the right databases.

`DB`	Statistic results database name
`COMMONDB`	Meta data Database name
`HOST`	MySQL database host name
`CLUSTER`	Name of the cluster node to use

You can fine-tune the number of threads to be used for any program that can make use of threads, like for example GEM. There is also a setting for the amount of memory to use for the Flux.

`THREADS`	Number of threads to use
`FLUXMEM`	Configures the memory used by the Flux. The default value is `16G`

The mapper and the number of mismatches can be set.

`MAPPER`	This currently has to be set to the value `GEM`
`MISMATCHES`	Number of mismatches for the mapper

The genome and annotation files need to be specified.

`GENOMESEQ`	Genome file
`ANNOTATION`	Annotation file

Preprocessing the reads should be done on the fly. The most common preprocessing step is trimming, so there is one setting for the trim length. You can also specify your own preprocessing script.

`PREPROCESS_TRIM_LENGTH`	A preprocessing step that trims the reads by the given nucleotide length
`PREPROCESS`	Path to a custom script used for preprocessing each of the read files before anything else

You can customize the way the recursive mapping is done, as well as how the postprocessing is done on some files.

MIN_RECURSIVE_MAPPING_TRIM_LENGTH	Tunes the minimum length to which a read will be trimmed during the recursive mapping.
MAXINTRONLENGTH	Sets the maximum length of splits allowed during the postprocessing of the files generated by `gem-2-sam` removing the noise. The default is set to `50k`, which is reasonable in mammals, however different species may require different settings. Setting it to `0` will remove this filter.

CHAPTER

FIVE

EXAMPLE PROFILE CONFIGURATION

Then we need to define the pipeline runs in:

```
profiles/MyProject/db.cfg
```

This is the content of the `db.cfg` file:

```
[runs]
parts = TestRun

[pipeline]
TEMPLATE   = ${buildout:directory}/src/pipeline/template3.0.txt
PROJECTID  = Test
DB         = Test_RNAseqPipeline
COMMONDB   = Test_RNAseqPipelineCommon
THREADS    = 8
MAPPER     = GEM
MISMATCHES = 2
CLUSTER    = mem_6
ANNOTATION = ${buildout:directory}/src/testdata/H.sapiens.EnsEMBL.55.test.gtf
GENOMESEQ  = ${buildout:directory}/src/testdata/H.sapiens.genome.hg19.test.fa

[TestRun]
recipe=grape.recipe.pipeline
accession = TestRun
```

Now that we have the accessions and profiles defined, we can go to our project folder and define the `buildout.cfg` that will produce our Grape pipelines:

```
pipelines/Test/buildout.cfg
```

The `buildout.cfg` should look like this:

```
[buildout]
extends = ../dependencies.cfg
          ../../accessions/Test/db.cfg
          ../../profiles/Test/db.cfg
```

There are pointers to the accession and profile. The dependencies file takes care of installing all the dependencies, like overlap, flux, gem, and the Grape pipeline.

Contents:

CHAPTER

SIX

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*