Grapevine User Manual v0.0.1

1. Overview

On a high level, Grapevine is an abstraction layer that converts any syntax into any other syntax. The translation from *source* to *target* is defined by *grammars*. A grammar is a set of rules that determine valid input, together with instructions as to how to map it to valid output.

In context of bioinformatics, Grapevine allows for easily implementing a workflow or pipeline that contains multiple processing steps. The input data is specified in a table that describes file locations and other attributes, which allows for processing data in batches.

Conceptually, Grapevine is designed to separate functionality such that components can be replaced without affecting the overall workflow. For example, the choice of a read aligner is defined by the grammar, so simple replacing or changing the grammar can switch from one program to another, given that the input and output files adhere to the same format (fastq in, sam or bam out).

For processing input with metadata, Grapevine takes a specified labeled table, from which variables can be accessed in the script. The number of output scripts is the number of data rows in this table. Together with a header file, the workflow can be configured so that the script does not change, while the header and table provide all information for the current run.

2. Setting up a Grapevine project

A Grapevine project typically consist of four components:

1. The main script
2. The data description table
3. The external header (optional)
4. A set of pre-defined or custom grammars

2.1 The main script

The main script first contains instructions for setting up the environment. If the target language is e.g. bash, and assuming SLURM as the scheduler, the first lines of the script might look like this:

```
#!/bin/bash -l

#SBATCH -A myProject
#SBATCH -p core
#SBATCH -n 16
#SBATCH -t 6:00:00

module load bioinfo-tools
export PATH=/home/manfredg/Software/minute-chip/bin/:$PATH
```

Next, the script defines what grammars to load, following this sytax:

```
>grammar = workflow/grammars/demux.gram
>grammar = workflow/grammars/mapping.gram
```

Variables are defined as follows:

```
@barcode = TGTCCAAT
```

Commands are then given in order following the syntax that is define by the grammar:

```
demultiplex folder first_run file base ONE barcode @barcode sample sample1
```

Where return variables can be collected like this:

```
demultiplex folder first_run file base ONE barcode @barcode sample sample1 > @out1 @out2
```

In this example, running Grapevine will result in the following script:

```
#!/bin/bash -l

#SBATCH -A myProject
#SBATCH -p core
#SBATCH -n 16
#SBATCH -t 6:00:00

module load bioinfo-tools
export PATH=/home/manfredg/Software/minute-chip/bin/:$PATH

>grammar = workflow/grammars/demux.gram
>grammar = workflow/grammars/mapping.gram

@barcode = TGTCCAAT

DeMuxFastq  -i1 ONE_R1.fastq.gz -i2 ONE_R2.fastq.gz -o1 first_run/ONE_sample1_R1.demux.fastq -o2
first_run/ONE_sample1_R2.demux.fastq -b TGTCCAAT -mis 1 -off 6
```

Where the variables @out1 and @out2 are set to “first_run/ONE_sample1_R1.demux.fastq” and “first_run/ONE_sample1_R2.demux.fastq” respectively, storing the locations of the output files that can then be passed on to subsequent steps.

Note that the script can contain any instructions in the target language, which will no be modified. This allows for rapid prototyping and debugging.

2.2 The data description table

To process multiple samples, a table can be specified in the main script or header:

```
@table = sampledata/table.txt
```

The table is blankspace (blank or tab) delimited with column headers in the first row, followed by entries for each sample or data set, e.g.:

```
analysis  sample       folder          filebase  barcode     taglen   description
H33       H33TAG_ctrl  20170828AA_MC   H3        CTACCAGG    6        Control
H33       H33TAG_P1    20170828AA_MC   H3        CATGCTTA    6        Experiment
H33       H33TAG_P2    20170828AA_MC   H3        GCACATCT    8        Experiment
H33       H33TAG_P3    20170828AA_MC   H3        AGCAATTC    7        Experiment
```

In the script or header, the entries can be directly accessed via @table. and the header names:

```
demultiplex folder @table.folder file base @table.filebase barcode @table.barcode sample
@table.sample
```

For each row in the table, Grapevine generates an executable script by filling in the value tables.

2.3 The external header

The header file will be included in the script, follows the same syntax, and allows for specifying a particular data set or options for processing the data, e.g. the variables:

```
@table = sampledata/table15.txt
@ref = /data1/common/references/mm10/mm10
```

might preferably be given in the header to specify a particular run that is processed against a particular database or genome reference.

2.4 A set of pre-defined or custom grammars

Grapevine uses the JSGF grammar format with particular extensions. It supports recursion as well as wild cards. The grammar header should specify the version and name:

```
#JSGF V1.0 ISO8859-1 en;
grammar com.test.dedup;
```

Grammars can import other grammars and access any public rule, e.g.:

```
import <com.test.digits.*>
```

Note that the depth of inclusion is currently limited to 1.

The grammar is a set of left hand rules (LHR) and right hand rules (RHR), specified as:

```
public <top> = process <dedup> | ignore;
```

terminated by a semicolon, and using the vertical bar to specify an OR relation. The ‘public’ keyword specifies that this rule may be accessed from other grammars. Private rules omit the ‘public’ keyword. There is no limitation on how deeply nested rules can be.

In the execution tags, delimited by curly brackets, variables are processed:

```
<file1> = in1 {this.in1 = “input1”}
```

where values are either specified explicitly, or via ‘*’, which returns the input:

```
<file1> = in1 {this.R1 = *}
```

Multiple statements within an execution tag are separated by semicolons. Wildcards are specified via the ‘%’ symbol, e.g.:

```
<file2> = in2 <wildcard> {this.R2 = this.wc; this.param = “in2”};

<wildcard> = % {this.wc = $};
```

Note that Grapevine will supply a set of grammars for common bioinformtics tools, with different sets of input syntax and supporting different target languages, such as bash and python.