Genomon Pipeline Development Guideline

2015/06/5 version 0.2

1. Source code structure

```
|-- db
|-- helpers
|-- perl
|-- python
|-- R
|-- resource
|-- samples
`-- shell
```

- perl, python, R, shell ディレクトリには、パイプラインで実際に使用されるスクリプトが配置されています。
- samples ディレクトリには、YAML ファイルのサンプルが配置されています。
- resource ディレクトリには、パイプラインで実行されるシェルスクリプトが、パイソンの文字列の形で格納されています。またその他の、文字リソースとして使用されている文字列が格納されています。
- helpers ディレクトリには、パイソンのプログラムが格納されています。

解析に使用するスクリプトを新たに加える場合は、このディレクトリにファイルを配置し、resource/genomon_rc.py の script_files に記述を加えてください。genomon_rc.py の script_files は、実行時に必要なスクリプトを result/.../script ディレクトリにコピーするファイルのリストを記述しています。

2. YAML files

サンプル YAML ファイルは、Github の Genomon の samples の中に、または、下記にあります。

/home/eigos/GenomonProj/test or Github:Genomon/samples

２つのタイプのファイルがあり、それぞれ別々に作成して指定することができます。

2.1. job description yaml file

Job description file には、解析のデータ、解析方法、などを記述します。

RNA-Seq データ解析の例）

RNA-Seq の例）

```
project_root:           /home/eigos/Data/GenomonProj/genomon
sample_name:            small_rna
```

```
sample_date:            20150409
analysis_date:          today

input_file_dir:         /home/eigos/Data/GenomonProj/input/small
input_file_type:        single_fastq
file_name:              d_100_*.fastq.bz2
file_ext:               .fastq.bz2
fastq_filter:           False # True or False

directory_permission:       owner   # owner, group, all

tasks:
                    RNA:
                        - tophat2
                        - cufflinks
                        - cummeRbund

#
#   HGC Shirokane super computer specific parameters
#
cmd_options:
            tophat2:              '-l ljob,s_vmem=2G,mem_req=2'
            cufflinks:            '-l ljob,s_vmem=2G,mem_req=2'
            cummeRbund:           '-l ljob,s_vmem=2G,mem_req=2'
qsub_cmd: "qsub -sync yes -now no {cmd_options} {cmd}"
drmaa_native: "-shell y {cmd_options}"
```

新たに機能を加える場合は、

1) 入力ファイルのフォーマットを指定。

2) 'tasks:'に新たにパイプラインの機能を追加する。

3) 'cmd_options:'に必要なメモリ量を指定する。

2.2. analysis parameter yaml file

Analysis parameter file には、解析を実行する時のパラメーターを指定します。指定されたパラメーターは、解析のスクリプトに渡されます。

パラメータ指定の例）

```
fisher_mutation_call:
    min_depth:              9
    max_indel:              2
    max_distance:           5
    base_quality:           15
    map_quality:            30
    mismatch_rate:          0.07
```

上記のように、job description yaml file に指定した、それぞれの処理でのパラメータを指定できます。

実際にパイプラインの中で値を取得するには、下記のように utility function を使用します。

```
   Geno.job.get_param( 'fisher_mutation_call', 'max_indel' )
```

2.3. system configuration file

```
[REFERENCE]
ref_fasta     = /home/w3varann/database/hg19/hg19.fa
[SOFTWARE]
python        = /usr/local/package/python2.7/currrent/bin/python
bwa           = /home/w3varann/tools/bwa-0.7.8/bwa
[ENV]
# biobambam needs libmaus library. libmaus_PATH is going to be
added in LD_LIBRARY_PATH.
libmaus_PATH  = /home/w3varann/tools/libmaus/lib
```

現在、３つのセクション、REFERENCE, SOFTWARE, ENV があります。

3. Utility functions

3.1. パラメーター取得の関数

1) Job description ファイルからのデータの取得

Geno.job.get_job を使ってデータを取得します。上記の RNA-Seq の例だと `input_file_type` を指定すると、`'single_fastq'` を返します。

例)

```
Geno.job.get_job( 'input_file_type' )  # returns 'single_fastq'.
```

2) Analysis parameter ファイルからのデータの

Geno.job.get_param を使ってデータを取得します。上記の RNA-Seq の例だと,
'fisher_mutation_call' の処理の 'min_depth' を指定すると、9 を返します。

例)

```
Geno.job.get_param( 'fisher_mutation_call', 'min_depth' ) #
returns 9.
```

3) System configuration ファイルからのデータの取得

Geno.conf.get を使ってデータを取得します。上記の RNA-Seq の例だと、'SOFTWARE'の 'bwa'を指定すると、下記にように system configuration ファイルに指定してある bwa のパスを返します。

例)

```
Geno.conf.get( 'SOFTWARE', 'bwa' ) # returns
'/home/w3varann/tools/bwa-0.7.8/bwa'
```

3.2. ジョブ実行の関数

1) single job の実行

Geno.RT.runtask 関数に、実行するシェルスクリプトの名前と、job description ファイルに指定してある、cmd_options (ジョブを実行する時のオプション。メモリー量、キューなど）を指定します。

```
Geno.RT.runtask(
   shell_script_full_path,
   Geno.job.get_job( 'cmd_options' )[ function_name ] )
```

2) array job の実行

Geno.RT.run_array_job 関数に、上記の２つの値と、アレイに関する値、id_start, id_end, id_step を指定します。id_step のデフォルト値は、1 に設定してあるので、指定しなくても動作します。

```
Geno.RT.run_arrayjob(
   shell_script_full_path,
   Geno.job.get_job( 'cmd_options' )[ function_name ],
   id_start = 1,
   id_end = 3 )
```

3.3. 実行結果の保存関数

1) 処理を実行した結果のフラグファイルの作成

Geno.status.save_status 関数に、実行した処理（例えば、tophat2、cufflinks など）、出力ファイル名、返り値を指定して、実行すると、config ディレクトリにフラグファイルを作成します。

```
Geno.status.save_status( function_name, output_file, return_code )
```

結果。初めの 3 ファイルは、パイプラインを実行した時の、job description ファイル、analysis parameter ファイル、system configuration ファイルです。残りの２ファイルが、bwa mem を実行してた時、結果ファイル g_g_100_1_XXX.bam を出力した時の、返り値が０のフラグファイルです。

```
genomon_20150603_1610_214891_param.yaml
genomon_20150603_1610_214891_job.yaml
genomon_20150603_1610_214891.cfg
genomon_20150603_1610_214891_bwa_mem_g_g_100_1_r0
genomon_20150603_1610_214891_bwa_mem_h_h_100_1_r0
```

2) 処理を実行した結果のフラグファイルのチェック

次回パイプラインを実行する時に、実行した処理が正常終了しているかをチェックする関数です。

上記の処理で作成されたファイルを見て、返り値を返します。ファイルが作成されていて、返り値が０になっていれば、正常終了です。

```
Geno.status.check_exit_status( process_name, output_file )
```

3.4. シェルスクリプトの作成

RNA-Seq tophat2 の例）

下記の例は、解析の script ディレクトリに実行するシェルスクリプトを作成するものです。

ファイル名を作成し、ファイルをオープンし、リソースファイルに指定してある、tophat2 の文字列データに必要なパラメーターを指定して、スクリプトを作成し、セーブします。

```
        shell_script_full_path =
make_script_file_name( function_name, Geno )
        shell_script_file = open( shell_script_full_path, 'w' )
        shell_script_file.write( rna_res.tophat2.format(
                                        log = Geno.dir[ 'log' ],
                                        ref_fa =
Geno.conf.get( 'REFERENCE', 'ref_fasta' ),
                                        input_fastq = input_file,
                                        output_file = output_file,
                                        ref_gtf =
Geno.conf.get( 'REFERENCE', 'ref_gtf' ),
                                        bowtie2_database =
Geno.conf.get( 'REFERENCE', 'bowtie2_db' ),
                                        bowtie_path = bowtie_path,
                                        tophat2 =
Geno.conf.get( 'SOFTWARE', 'tophat2' )
                                    )
                                )
        shell_script_file.close()
```

4. Execution and Debug

実行方法 1 ）

```
run.sh   [Job description YAML file]\
         [Analysis parameter YAML file]
```

実行方法 2）python debugger を使う。

```
run.sh   [Job description YAML file]\
         [Analysis parameter YAML file]\
         '-m pdb'
```

パラメーター)

genomon.py に指定できるパラメーターです。

```
$ >python ./genomon.py
```

```
usage: genomon.py [-h] [--verbose [VERBOSE]] [--version] [-L FILE]
                  [-T JOBNAME] [-j N] [--use_threads] [-n]
                  [--touch_files_only] [--recreate_database]
                  [--checksum_file_name FILE] [--flowchart FILE]
                  [--key_legend_in_graph] [--draw_graph_horizontally]
                  [--flowchart_format FORMAT] [--forced_tasks JOBNAME]
                  [-s CONFIG_FILE] [-f JOB_FILE] [-p PARAM_FILE] [-m] [-d]
                  [-l]

Genome Analysis Pipeline

optional arguments:
  -h, --help            show this help message and exit

Common options:
  --verbose [VERBOSE], -v [VERBOSE]
                        Print more verbose messages for each additional
                        verbose level.
  --version             show program's version number and exit
  -L FILE, --log_file FILE
                        Name and path of log file

pipeline arguments:
  -T JOBNAME, --target_tasks JOBNAME
                        Target task(s) of pipeline.
  -j N, --jobs N        Allow N jobs (commands) to run simultaneously.
  --use_threads         Use multiple threads rather than processes. Needs
                        --jobs N with N > 1
  -n, --just_print      Don't actually run any commands; just print the
                        pipeline.
  --touch_files_only    Don't actually run the pipeline; just 'touch' the
                        output for each task to make them appear up to date.
  --recreate_database   Don't actually run the pipeline; just recreate the
                        checksum database.
  --checksum_file_name FILE
                        Path of the checksum file.
  --flowchart FILE      Don't run any commands; just print pipeline as a
                        flowchart.
  --key_legend_in_graph
                        Print out legend and key for dependency graph.
  --draw_graph_horizontally
                        Draw horizontal dependency graph.
  --flowchart_format FORMAT
                        format of dependency graph file. Can be 'pdf', 'svg',
                        'svgz' (Structured Vector Graphics), 'pdf', 'png'
                        'jpg' (bitmap graphics) etc
  --forced_tasks JOBNAME
                        Task(s) which will be included even if they are up to
                        date.

genomon:
  Genomon options

  -s CONFIG_FILE, --config_file CONFIG_FILE
                        Genomon pipeline configuration file
  -f JOB_FILE, --job_file JOB_FILE
                        Genomon pipeline job configuration file
  -p PARAM_FILE, --param_file PARAM_FILE
                        Genomon pipeline analysis parameter file
  -m, --mpi             Use MPI job submission
  -d, --drmaa           Use DRMAA job submission
  -l, --abpath          Use absolute path in scripts
```

genomon.py を実行するには、下記の環境変数を指定しなければなりません。

```
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/package/python2.7/curr
ent/lib
export
PYTHONPATH=$PYTHONPATH:/home/w3varann/.local/lib/python2.7/site-
packages
```

run.sh で使用されているパラメーターは、下記に示したとうりです。--config_file、--job_file、--param_file は、必ず指定しなければなりません。

```
           --config_file
/home/eigos/Data/GenomonProj/test/genomon.cfg \
           --job_file $1 \
           --param_file $2 \
           --jobs 10 \
           --verbose 10
```

- --drmaa/-d: デフォルトでは、qsub によってジョブを投入します。DRMAA(Distributed Resource Management Application API を使用して、ジョブを投入する場合は、--drmaa を指定してください。
- --abpath/-l: デフォルトでは、project_root からの相対パスを使って解析の処理スクリプトを作成します。相対パスだと、project_root ごと他の場所にコピーしても、処理スクリプトを再実行することができます。
- --jobs/-j: 並列処理の並列度を指定できます。