pyWrairLib Documentation

Tyghe Vallard

May 2013

Contents

1 The Pipeline

1.1 Introduction

The concept of The Pipeline is quite simple. Execute one script after another performing one task after another on the data until it is processed. Because NGS Data is so large, it is best to run most of these scripts on a Multiprocessor/Multicore system. If you want to know how many CPUs and Cores you have in linux just issue the following command

```
cat /proc/cpuinfo
```

You will have to manually look through this specifically looking for the physical id and cpu cores

1.2 Pipline Pieces

The Pipeline is composed of two parts:

1. Data Setup
2. Analysis

1.3 Available Scripts

These are the available scripts that drive the pipeline. The full documentation for each of the scripts is detailed at the bottom of the document.

1.3.1 Analysis

- genSummary.sh
  - Runs summary scripts to generate compiled summary output for GsMapper projects inside a directory
- mapSamples.py
  - Creates and runs gsmapper projects based on a RunFile
- mapSummary.py
  - Creates AllRefStatus.xls file which gives mapping depth and coverage percentage for each reference that was used in the reference mappings
- genallcontigs.py
  - Creates a directory containing all GsMapper project's 454AllContigs.fna+.qual consolidated into a single directory. Each AllContigs fasta+qual file is merged into a single .fastq file named after the GsMapper project directory from which it originated.
- allcontig_to_allsample.py
  - Merges a GsMapper project's 454AllContigs.fna and 454AllContigs.qual file into a single .fastq
- genallrefstatus.sh
  - Legacy shell script that merges together all found 454RefStatus.txt files into a single output stream/file
- variant_lookup
  - Script to aid in the lookup of variant information

1.3.2 Data

- demultiplex
  - Demultiplexes all found sff files inside of a given sff directory. Also renames the generated files utilizing a given runfile.
- link_reads
  - Links all valid reads found in a given directory into the NGSData's ReadsBySample directory
- sanger_to_fastq
  - Helper script to convert sanger .ab1 sequence files to .fastq files. Useful as Newbler only accepts .sff or .fastq as input sequence format.

1.3.3 Misc/Undocumented/InDevelopment

- bestblast
- entrez_helper
- reads_for_contig
- refrename.py
- splitsff

1.4 Running the Pipeline

1.5 Data Setup

- Copy the NGS data from the instrument into its appropriate location under the NGSData/RawReads directory

```
cp -R <path to ngsdata on usb drive or wherever> <path to NGSData>/RawReads/<sequencer type>/
```

- Create a meta directory for that run

```
mkdir <path to NGSData>RawReads/<sequencer type>/<run date of sequencer>/meta
```

- Create Primer and Ref directories and copy RunFile into meta directory

```
for adir in Primer Ref; \
do \
    mkdir <path to NGSData>RawReads/<sequencer type>/<run date of sequencer>/meta/${adir}; \
done
mv <path to NGSData>RawReads/<sequencer type>/<run date of sequencer>/<runfile>.txt <path to NGSData>RawReads/<sequencer type
    >/<run date of sequencer>/meta/
cp <path to primer fasta files> <path to NGSData>RawReads/<sequencer type>/<run date of sequencer>/meta/Primer/
```

- Link the Signal processing directory from the RawReads sequencer directory into NGSData/ReadData

```
ln -s <path to NGSData>RawReads/<sequencer type>/<run date of sequencer>/D\_*signalProcessing* <path to NGSData>/ReadData/<
    sequencer type>/
```

- Change directory to the Signal Processing Directory for the run you want to demultiplex

```
cd <path to NGSData>/ReadData/<sequencer type>/<signal processing directory>
```

- Run the demultiplex script to demultiplex the raw sff files and rename them according to the runfile

```
demultiplex -s sff/ -r <path to runfile>
```

- Link the reads from the demultiplexed directory into the NGSData/ReadsBySample directory

```
link_reads
```

After this is completed all the data for each sample will be located under the NGSData/ReadsBySample directory under that sample's name This is important because mapSamples.py looks there for read data

1.6 Analysis

- Change directory to the base of your Analysis data

```
cd <path to Analysis>
```

- Create a new analysis directory and change directory to it. Typically named after the NGS Sequencer run you wish to analyze

```
mkdir <YYYY_MM_DD>
cd <YYYY_MM_DD>
```

- Link in the meta directory for the NGS Run

```
ln -s <path to NGSData>RawReads/<sequencer type>/<run date of sequencer>/meta .
```

- Copy any needed references into the meta/Ref folder
- Create a directory for the analysis with whatever name you want with a workfile directory to place summarys and RunFile

```
mkdir myanalysis && mkdir myanalysis/workfile
```

- Copy Runfile from meta directory to workfile directory

```
cp meta/Runfile.txt myanalysis/workfile
```

- Edit the copied runfile inside of the workfile directory and set references for each of the samples. You may specify a directory of references instead of individual reference files to include all *.fasta files inside of that directory.

- Once the runfile is copied you can run the mapping for all samples inside of the RunFile that are not commented out(See more about RunFiles below). Make sure you have changed directory to the analysis directory you created.
  - Be patient if there are a lot of samples to map.

```
cd myanalysis
mapSamples.py workfile/Runfile.txt
```

- When mapSamples.py is finished you can generate summeries with the genSummary.sh script

```
genSummary.sh
```

2 Config Files

2.1 settings.cfg

This is the main configuration file that controls how all of the libraries and scripts intereact in the various pyWrairLib modules. When pyWrairLib is installed this file is copied into a directory called config inside of the PYTHONHOME environmental variable *You can see where this is by issuing*

```
echo $PYTHONHOME
```

If nothing is displayed then you need to locate your system's default python home

2.2 MidParse.conf

This file is used by the roche software to demultiplex sff files. The contents of the file contain a mapping between a name and a barcode sequence. The structure of the file is as follows

```
MID
{
    mid = "MIDNAME1", "BARCODESEQUENCE", MISMATCHTOLERANCE, "OPTIONAL 3' Trim Sequence";
    mid = "MIDNAME2", "BARCODESEQUENCE", MISMATCHTOLERANCE, "OPTIONAL 3' Trim Sequence";
}
```

More information on this file can be found in the Roche documentation Part C under MIDConfig.parse. An example file that can be used is included in the config directory

2.3 RunFile

Run files are simply a file that links a samplename to the following information about that sample

- Region number
- samplename
- virus name
  - Should conform to a valid wrair virus name(More details later on that)
- barcode/midkey name
  - Has to be a valid midkey name inside of the Midparse.conf file(for sfffile to demultiplex with using –mcf option)
- mismatch tolerance
- reference directory/file to use to map to
  - Can be a directory of reference files or a single reference
- unique sampleid
  - Deprecated, just use the same as you put in the samplename column
- primerfile location
  - Fasta file of primers to trim off

The following RunFile templates utilize Python Template Strings You can read more about them at:
http://docs.python.org/2/library/string.html#template-strings
Basically replace anything that has a dollar sign in front of it with a value

2.3.1 RunFile Header

```
# $platform sample list
# $numregions Region $type
# Run File ID: $date.$id
!Region Sample_name      Genotype        MIDKey_name     Mismatch_tolerance      Reference_genome_location       Unique_sample_id
        Primers
```

- The header line (starts with a !) is tab separated
- $platform would be replaced by Roche454 or IonTorrent
- $date needs to be a valid date string that is in one of the following formats
  - DDMMYYYY
  - DD_MM_YYYY
  - YYYY_MM_DD
- $id is anything that does not contain a space character

2.3.2 RunFile Sample Line

This line is also tab separated

```
$region $sample $virus  $midkey $mismatch       $reference      $sample $primer
```

2.3.3 Sample RunFile

Here is an example RunFile with 2 samples
Sample D1_FST2432 is in region 1 and has barcode IX001 from the MidParse.conf file
Sample D1_FST2410 is in region 2 and has barcode IX002 from the MidParse.conf file

```
# IonExpress sample list
# 2 Region PTP
# Run File ID: 04192013.PGM.CPTLin
!Region Sample_name      Genotype        MIDKey_name     Mismatch_tolerance      Reference_genome_location       Unique_sample_id        Primers
1       D1_FST2432       Den1     IX001  0       Analysis/PipelineRuns/2013_04_19/Ref/Den1       D1_FST2432      NGSData/RawData/IonTorrent/2013_04_19/meta/Primer/Den1.fna
2       D1_FST2410       Den1     IX002  0       Analysis/PipelineRuns/2013_04_19/Ref/Den1       D1_FST2410      NGSData/RawData/IonTorrent/2013_04_19/meta/Primer/Den1.fna
```

3 Libraries

This section lists all of the libraries that make up pyWrairLib
Each library may contain scripts as well which are listed with their usage

3.1 wrairdata

3.1.1 sanger_to_fastq

Convert .ab1 files into various formats(by default fastq) This script is under development and should be considered as such That being said, it seems to work fine giving it the -d and -t options together

Usage

```
usage: sanger_to_fastq [-h] [-s SANGERFILE] [-d SANGERDIR] [-o OUTPUTFILE]
                       [-t OUTPUTTYPE]

Convert sanger .ab1 file to another type

optional arguments:
  -h, --help            show this help message and exit
  -s SANGERFILE, --sanger-file SANGERFILE
                        Sanger file to convert
  -d SANGERDIR, --sanger-dir SANGERDIR
                        Directory containing sanger files
  -o OUTPUTFILE, --output-file OUTPUTFILE
                        The output filename. Omit an extension if you use the
                        -t fasta+qual option.
  -t OUTPUTTYPE, --output-type OUTPUTTYPE
                        The output file type. Check out
                        http://biopython.org/wiki/SeqIO for options. Use
                        fasta+qual to get both fasta and qual files
```

3.1.2 demultiplex

While there are many options for this command, only the -r and -s options are typically used.

- -r Specifies the location of the runfile so it knows how to map sample names with what barcode/region they are from
- -s Specifies the location of the sff directory that contains the multiplexed sff files to demultiplex

Usage

```
usage: demultiplex [-h] [-d PROCDIR] [-r RUNFILE] [-s SFFDIR] [-o OUTPUTDIR]
                   [--mcf MIDPARSEFILE] [--sfffilecmd SFFFILECMD] [--rename]
                   [--demultiplex]

optional arguments:
  -h, --help            show this help message and exit
  -d PROCDIR, --image-processing-dir PROCDIR
                        Image processing directory path
  -r RUNFILE, --runfile RUNFILE
                        Path to the Runfile
  -s SFFDIR, --sff-dir SFFDIR
                        Path to directory containing sff files
  -o OUTPUTDIR, --output-dir OUTPUTDIR
                        Output directory path[Default: demultiplexed/]
  --mcf MIDPARSEFILE    Midkey config parse file[Default:
                        NGSData/MidParse.conf]
  --sfffilecmd SFFFILECMD
                        Path to sfffile command[Default:
                        bin/sfffile]
  --rename              Only rename already demultiplexed sff files
  --demultiplex         Only demultiplex. Don't rename
```

- At this time the -d option is in testing and unstable

3.1.3 link_reads

link_reads is a simple helper script to link NGS reads to the NGSData's ReadsBySample directory link_reads by default looks to see if there is a demultiplexed directory in the current directory you are in otherwise defaults to just the current directory you are in or you can manually set the directory by using the -d option

Typically link_reads is run right after running demultiplex

Usage

```
usage: link_reads [-h] [-d INPUTDIR] [-c CONFIGPATH]

optional arguments:
  -h, --help            show this help message and exit
  -d INPUTDIR, --demultiplexed-dir INPUTDIR
                        Directory containing read data[Default:
                        demultiplexed]
  -c CONFIGPATH, --config CONFIGPATH
                        Config file to use
```

3.2 wrairlib

3.3 wrairnaming

3.4 wrairanalysis

3.4.1 mapSamples.py

mapSamples.py is used to map samples to a reference genome using the Roche Newbler mapper. It derives all of its parameters from a RunFile. It will create GsMapper project directories inside of the directory that you execute it from. It is usually best to create a directory to run this command from within.

Usage

```
usage: mapSamples.py [-h] [-c CONFIGPATH] runfile

positional arguments:
  runfile               Runfile path to use

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIGPATH, --config CONFIGPATH
                        Config file to use
```

3.4.2 genSummary.sh

This is a wrapper shell script that runs post mapping summary scripts. It doesn't generate any output or output files itself, but runs scripts that do generate output. It runs all scripts on all GsMapper directories found in the current directory it is executed from. It is easy to run after mapSamples.py is run since mapSamples.py creates GsMapper projects in a directory and then you can just run genSummary.sh after that to get a more compiled look at all of the projects status **Currently runs:**

- Gap analysis
  - Compiles the output of NGSCoverage's gapsformids.py into images that are separated by each virus's segment inside of Gaps/Segments
- genallcontigs.py
- mapSummary.py
- SequenceExtraction's Summary -table

Usage

```
genSummary.sh
```

You must run this from within the same directory that mapSamples.py was run

3.4.3 mapSummary.py

Given a directory containing gsMapper projects, this script will find all 454RefStatus.txt files and compiles a single Excel spreadsheet from them for all references used across all projects. You can also specify a reference file to 'target' so that only the references inside that file will be listed in the output excel file.

Usage

```
mapSummary.py -d <projectdir> [-r <reference>] [-o <outputfile>]
```

- projectdir can by any directory that directly contains GsMapper projects. Usually this directory is the same directory that mapSamples.py was executed from.
- reference is optional and should be one of the reference files that was used in the mapping. This basically filters out all references so that only the reference you specify will be listed in the output
- outputfile is optional and specifies where to write the resulting Excel file. The default is in the current directory with the name AllRefStatus.xls

3.4.4 genallcontigs.py

Searches inside a given directory for any gsMapper directories For every gsMapper directory found, it gathers all contigs from the 454AllContigs.fna and writes them to a file named after the project directory inside of the given output directory. The output is essentially a single directory containing Merged 454AllContigs.fna and .qual files into a .fastq file for each gsMapper directory

Usage

```
genallcontigs.py -d <projectdir> [-o <outputdir>]
```

- projectdir can by any directory that directly contains GsMapper projects. Usually this directory is the same directory that mapSamples.py was executed from.
- outputdir is optional and defines what directory to place the results in. By default it is FastaContigs inside the current directory.

3.4.5 allcontig_to_allsample.py

Given a single gsMapper project directory, merges the 454AllContigs.fna and 454AllContigs.qual files into a single .fastq file Output is by default sent to STDOUT(screen) but can be set to a file by using the -o option

Usage

```
allcontig_to_allsample.py -p <gsproject> [-o <outputfile>]
```

- gsproject is the path to a single Gs Project
- outputfile is optional and should be a file to write the resulting fastq to. By default it is written to the terminal

3.4.6 variant_lookup

Variant lookup looks up a position in a GsMapper alignment and displays the information about that variant You just have to provide the nucleotide position and optionally a unique portion of the reference you are looking for.

If you do not provide the reference it will display all references for that alignment

Usage

```
usage: variant_lookup [-h] [-d PDPATH] variant_pos [ref_name]

positional arguments:
  variant_pos           The variant position to display info for
  ref_name              The identifier to limit the info for. Default is to
                        show all identifiers

optional arguments:
  -h, --help            show this help message and exit
  -d PDPATH, --project_directory PDPATH
                        GsProject directory path[Default:Current working directory]
```

- variant_pos is the Nucleotide position of the alignment to look for the information for
- ref_name only has to be a portion of the reference you are looking for. This is simply a filter for all reference names in the alignment.

  ```
  Example:
    If the alignment has 3 references
     Reference_ABCD
     Reference_ACDE
     Reference_1
  ```

  - Specifying Reference as the ref_name argument would yield all 3 references
  - Specifying ABCD would only yield Reference_ABCD
  - Specifying Reference_A woudl yield Reference_ABCD and Reference_ACDE
- project_directory is only used if your current working directory is not a GsMapper project. It is easiest to invoke variant_lookup by first changing directory to the GsMapper project you are wanting to work on.