DECONVQC Processing Pipeline

Table of Contents

[2 Introduction 2](#_Toc460240306)

[3 Lab Pipeline 2](#_Toc460240307)

[3.1 Overview 2](#_Toc460240308)

[4 Compute Pipeline 2](#_Toc460240309)

[4.1 Overview 2](#_Toc460240310)

[4.2 Implementation 3](#_Toc460240311)

[5 HiSeq pipeline Usage 3](#_Toc460240312)

[5.1 Stage 1 – Generate and Q/C fastq data 3](#_Toc460240313)

[5.2 Stage 2 – GBS Q/C run using Tassel uneak and KGD 3](#_Toc460240314)

[5.3 Stage 3 – Move to online archive 4](#_Toc460240315)

[5.4 How-to 4](#_Toc460240316)

[5.4.1 Updating a key-file 4](#_Toc460240317)

[5.4.2 (RE-)Running a subset of samples Through GBS step 5](#_Toc460240318)

[5.4.3 Annotate a run or sample in the Database 6](#_Toc460240319)

[5.4.4 Extract one of the standard key files from the database 6](#_Toc460240320)

[5.4.5 Extract a Custom Keyfile from the Database - Example: A keyfile to include deer species, where the enzyme PSTI was used 7](#_Toc460240321)

[6 References 9](#_Toc460240322)

[7 Appendices \*\*\*\* Please Ignore This is Under DEVELOPMENT! \*\*\*\*\* 9](#_Toc460240323)

[7.1 Lab DataFlow 9](#_Toc460240324)

[7.1.1 Data Flow Overview 9](#_Toc460240325)

[7.1.2 Source and intermediate data-file examples 10](#_Toc460240326)

[7.1.3 Screen Shots of Key Genophyle Interactions 10](#_Toc460240327)

[7.1.4 Key-file Extract Stored Procedure Development 10](#_Toc460240328)

# Introduction

The development of database and compute support for the HiSeq based GBS initiative is necessarily fairly iterative, since every aspect of this work, from sample prep to quantitation to HiSeq run configuration to key-file formats and GBS analysis software selection, is subject to fairly frequent change and review as we find out which biological and technical processes and GBS algorithms work well and which don’t. Therefore in practice the software processes and procedures are “still maturing” and running these analyses is still somewhat “hands-on”. In this regard we try to follow a continuous improvement process, whereby for each run (usually 1 per week), we try to pick one data processing / software task or part of a task to be improved and automated, as part of processing that run.

# Lab Pipeline

## Overview

It is import to remember that the analysis of a sample in the Genomics / GBS lab typically involves multiple rounds of quantitation and decision making, with each round generally yielding finer quantitation than the last. Early rounds yield broad brush quantitation, for example DNA concentration, and this is used as part of decision making related to the configuration of later quantitation (for example plate layouts which involve segregating samples by DNA concentration).

Sequencing of a sample is generally the last round of quantitation as far as the wet lab is concerned though of course there is then a similar series of rounds of in-silico “quantitation” (which reads pass Q/C? which reads align to reference? etc. etc.), and decision branching. The overall data-flow is thus quite complex, with many potential branch points for a sample on its way through both the wet and in-silico lab processes, as decisions are made based on sample quantitation up to that point. Database support is therefore also quite complex with for example a sample occupying a series of plate positions on its way through the lab.

LIMS support is currently provided by the legacy in-house “Genophyle database”, in combination with a number of flat file based processes (for example related to key-files). This is an interim solution pending provision of updated LIMS and data warehouse support. A description of the data flows and procedures, and links to examples of input and output files, ~~is~~ will be set out in Appendix 7.1 below.

# Compute Pipeline

## Overview

The HiSeq compute pipeline starts with a completed run on the AgResearch hiseq zfs based storage system. This data is then processed by the pipeline in three stages:

1. Generate and Q/C fastq files. This runs bcl2fastq and some Q/C checks on that data, all written to working storage. This stage is not GBS-specific (i.e. does not require key-files etc.)
2. For GBS oriented sequencing (i.e. currently all runs on these machines), a Q/C GBS run is done on the sequence data from stage 1, using Tassel UNEAK and KGD. This stage also imports the run metadata into the postgres database, and updates key-files with the locations of fastq files from Stage 1
3. Move sequence data and metadata out of working storage and into long term on-line archive (includes reducing footprint by deletion of e.g. image / bcl files etc). Unlike most sequencing centres (which can delete sequence data from the system once uplifted), the AgResearch centre needs to accumulate all of the sequence data that is generated to support the main GBS use-case of the facility.

## Implementation

Each of the three stages is implemented as a top-level bash script, paired up with a make-file which is called by the script.

Stage 1: process\_hiseq1.0.sh (and process\_hiseq1.0.mk)

Stage 2: gbs\_hiseq1.0.sh (and gbs\_hiseq1.0.mk)

Stage 3: archive\_hiseq1.0.sh (and archive\_hiseq1.0.mk)

Each of the above top-level script and make-files may call a number of helper scripts. Each top-level script has a “dry-run” (-n) option which can be used to preview what processing will be done without actually doing it and a help (-h) option giving usage examples.

Basic run metadata such as sample sheet, GBS key-files are stored in a postgres database which is accessed by Stage 2. There are no database dependencies in stages 1 and 3. There is a web-interface to the database that can be used to browse and annotate runs, samples and key-files – however there is no dependency of the pipeline on the web-interface.

GBS Key-files are currently also maintained in a mercurial source-control repository.

Stage 1 is partly run on a compute cluster managed by condor. In all stages advantage is taken of make’s ability to launch multiple threads to process independent targets (e.g. make -j 8 etc). This means for example we can launch multiple concurrent GBS processing runs (stage 2) and file copy commands (stage 3), without having to explicitly code this.

All of the top level scripts and make invocations write verbose logging information to standard log files.

# HiSeq pipeline Usage

## Stage 1 – Generate and Q/C fastq data

In principle this would be started automatically via a cron job polling for a run-completion landmark file such as “RTAComplete.txt” in the run folder. Currently it is still started manually.

Example:

process\_hiseq1.0.sh -i -r 150515\_D00390\_0227\_BC6JPMANXX

(150515\_D00390\_0227\_BC6JPMANXX is the name of the run folder as generated by the Illumina sequencing run )

## Stage 2 – GBS Q/C run using Tassel uneak and KGD

As well as executing the GBS Q/C analysis, this stage also imports the run metadata into the database, and updates the relevant GBS key-files with the locations of fastq files from stage 1. Because key-file management processes are “not yet mature”, there are sometimes surprises – for example if we inadvertently change the headings or introduce non-ascii content when saving from Excel templates. Therefore this stage is executed in two steps: (1) the database import – run, review, clean up any problems, and then (2) launch the GBS run.

Database import example:

./gbs\_hiseq1.0.sh -i -t db\_update -r 160429\_D00390\_0252\_AC9CHFANXX > 160429\_D00390\_0252\_AC9CHFANXX.dbupdate 2>&1

GBS run example:

./gbs\_hiseq1.0.sh -i -r 160429\_D00390\_0252\_AC9CHFANXX

## Stage 3 – Move to online archive

This stage is not done synchronously with the other two stages - online archiving currently lags the other two stages by several months, though in future this will reduce to “several weeks”

This is done in three steps: (1) copy across the fastq and metadata files from working storage to online archive (2) adjust the soft links in the fastq link-farm, to point to the new online-archive locations. This stage also validates that the fastq copies are complete via comparing sequence counts (3) delete raw data (bcl, image files) and the working-storage copies of the fastq files.

This process is still maturing and is also still fairly hands-on, due to the importance of validating that the online archive fastq files are correct, and replicated off-site, before clean-up.

Run archiving example:

./archive\_hiseq1.0.sh -r 141217\_D00390\_0214\_BC4UEHACXX

Patch link-farm example:

./fix\_archived\_run\_fastq\_links.sh -v -r 141217\_D00390\_0214\_BC4UEHACXX -f C4UEHACXX 1>141217\_D00390\_0214\_BC4UEHACXX.relinkstdout 2>141217\_D00390\_0214\_BC4UEHACXX.relinkerr

Clean up:

This is not yet scripted - but will be scripted soon as part of continuous improvement

## How-to

### Updating a key-file

This can be done using the script importOrUpdateKeyfile.sh. For example:

|  |
| --- |
| # dry run  importOrUpdateKeyfile.sh –n -s SQ0223 -k SQ0223  # actual  importOrUpdateKeyfile.sh -s SQ0223 -k SQ0223  # you may need to run the updateFastqLocations.sh script after import as below |

Alternatively you could delete and import as below

In this example, a number of samples in SQ0223 were coded as deer when they should have been cattle. This did not need anything to be re-run, but important to correct the key-file in the database , so that future custom key-file queries pull out the correct information.

|  |
| --- |
| # delete the keyfile from the database  pwd  /dataset/hiseq/active/bin/hiseq\_pipeline/database  export GBS\_BIN=/dataset/hiseq/active/bin/hiseq\_pipeline  ./deleteKeyfile.sh -s SQ0223 -k SQ0223  # reimport the corrected keyfile. (the -f (“force”) option is needed because otherwise the script will warn that this key-file has already been imported and will not reimport it)  ./importKeyfile.sh -f -s SQ0223 -k SQ0223  # if the correct keyfile did not include the fastq locations, then need to update the newly imported key-file with these. (If it did include them , then no update will be done, as this update will only update the fastqlocation column if it is empty – i.e. in that case no harm done )  ./updateFastqLocations.sh -s SQ0223 -k SQ0223 -r 160531\_D00390\_0253\_AC9AB9ANXX -f C9AB9ANXX -l 4 |

Finally – the sample should be annotated using the database page for the sample – see e.g.

<http://agbrdf.agresearch.co.nz/cgi-bin/fetch.py?obid=SQ0223&context=default>

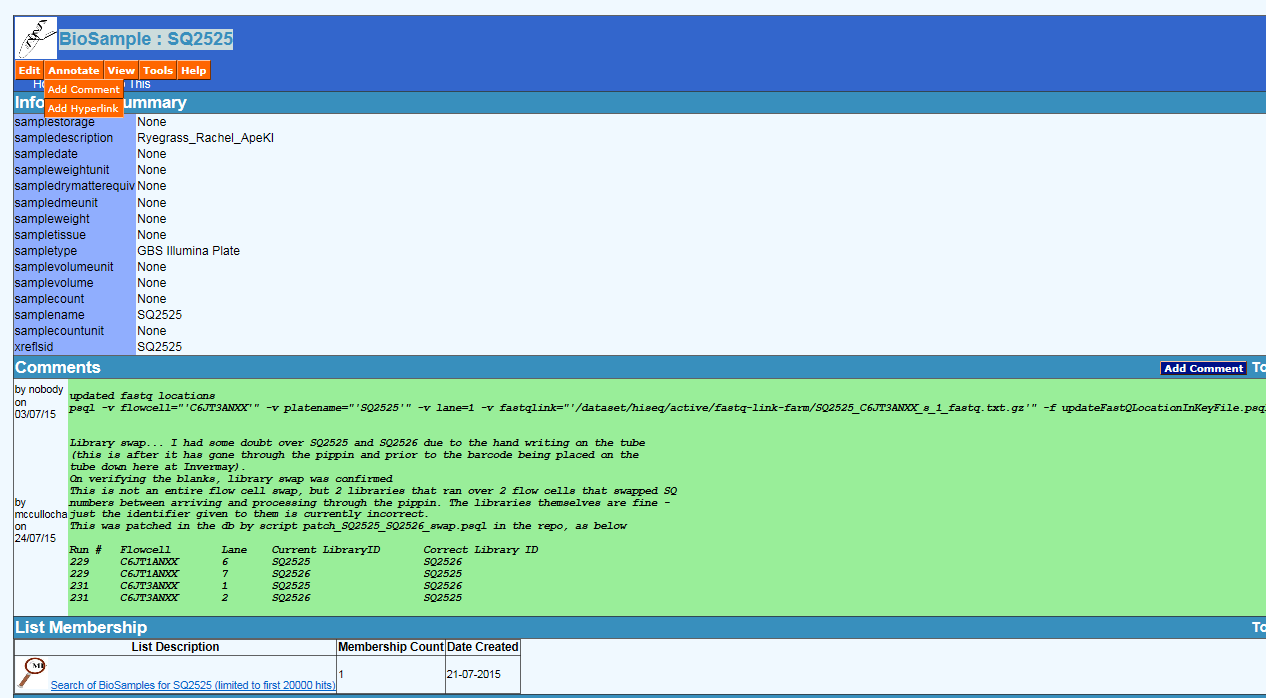
### (RE-)Running a subset of samples Through GBS step

This is usually associated with a correction of a keyfile (e.g. bar-code swap / mislabel) and so usually involves reloading the keyfile – the procedure for that is as above

|  |
| --- |
| # -- reload relevant keyfile(s) as above –  # save the existing GBS folder – the rerun will create a new one  mv /dataset/hiseq/scratch/postprocessing/160531\_D00390\_0254\_BC9CCNANXX.gbs /dataset/hiseq/scratch/postprocessing/160531\_D00390\_0254\_BC9CCNANXX.gbs.save  # rerun the sample required (will create a new 160531\_D00390\_0254\_BC9CCNANXX.gbs folder)  ./gbs\_hiseq1.0.sh -i -r 160531\_D00390\_0254\_BC9CCNANXX -s SQ0231  # assuming we want to keep the original version of that sample’s results for a while – make a copy  mv /dataset/hiseq/scratch/postprocessing/160531\_D00390\_0254\_BC9CCNANXX.gbs.save/SQ0231.processed\_sample /dataset/hiseq/scratch/postprocessing/160531\_D00390\_0254\_BC9CCNANXX.gbs.save/SQ0231.processed\_sample.bad  # move the corrected sample version into the saved folder with the other samples  mv /dataset/hiseq/scratch/postprocessing/160531\_D00390\_0254\_BC9CCNANXX.gbs/ SQ0231.processed\_sample /dataset/hiseq/scratch/postprocessing/160531\_D00390\_0254\_BC9CCNANXX.gbs.save/SQ0231.processed\_sample  # remove the now empty re-run folder  rmdir /dataset/hiseq/scratch/postprocessing/160531\_D00390\_0254\_BC9CCNANXX.gbs  # rename the save folder, which now contains both the corrected and original sample output  mv /dataset/hiseq/scratch/postprocessing/160531\_D00390\_0254\_BC9CCNANXX.gbs.save /dataset/hiseq/scratch/postprocessing/160531\_D00390\_0254\_BC9CCNANXX.gbs |

### Annotate a run or sample in the Database

* Open the [report of runs and samples](http://agbrdf.agresearch.co.nz/cgi-bin/fetch.py?obid=illumina_samplesheets_report1&context=execute&target=ob) at <http://agbrdf.agresearch.co.nz>
* To annotate a sample click on the sample link to open the page for the sample. On the sample page click “Annotate” and choose either “Add comment”, or “Add hyperlink”. The screen-shot below shows an example of annotating a sample. Usually you would add a comment, but in some cases it might be useful to link to another document, so you would add a hyperlink. You can add as many comments and hyperlinks as needed. Comment length is not restricted but overly long comments are probably hard to read.
* You can’t edit or delete comments. If you really need this contact an admin
* You can enter a single comment or hyperlink and have it appear against multiple samples/runs – if you need to do this, enter a comment and ask an administrator to link it to other records.
* To annotate a run , the process is exactly the same , but you do this from the web page from the run – i.e. click on the run hyperlink and then proceed as above.



### Extract one of the standard key files from the database

While in most cases the key-file staging area /dataset/hiseq/active/key-files contains up-to-date key-files, this is no longer a primary key-file repository, and key-files in this folder are not guaranteed to be in sync with the database. For each analysis it is best to extract a key-file specifically for that analysis from the database (this also ensures the analysis is reproducible – whereas e.g. referencing key-files directly from the staging area could be problematic as they sometimes change with further sequencing runs)

To extract an up-to-date key-file, you can use a command like the following:

|  |
| --- |
| /dataset/hiseq/active/bin/hiseq\_pipeline/database/listDBKeyfile.sh -s SQ2568 # tassel 3  /dataset/hiseq/active/bin/hiseq\_pipeline/database/listDBKeyfile.sh -s SQ2568 -v 5 # tassel 3 |

(Both these commands list the keyfile to stdout, so you can redirect to your own file)

### Extract a Custom Keyfile from the Database - Example: A keyfile to include deer species, where the enzyme PSTI was used

You can connect to the database hosting key-file information using the gbs user, password is *keyfiles*

Currently controlled vocabs are not enforced, so to design a query to pull out everything that is applicable you need to first check what terms have been used:

1. Check what species nomenclature is present

|  |
| --- |
| intrepid$ psql -U gbs -d agrbrdf -h invincible  Password for user gbs:  psql (8.4.20)  Type "help" for help.  agrbrdf=> select species,count(\*) from gbskeyfilefact group by species order by 1;  species | count  -----------------------+-------  | 1  Argentine\_stem\_weevil | 96  Atlantic\_salmon | 1728  Atlantic Salmon | 10853  Cattle | 586  Chinook\_salmon | 384  Chinook Salmon | 96  Crow | 87  Deer | 3147  Endophyte | 96  Fur\_seal | 88  Goat | 32  Greenshell\_mussel | 207  GreenShell Mussel | 192  Pea | 192  Pig | 4  Ryegrass | 8162  Salmon | 108  Sheep | 4543  Tuatara | 57  White clover | 1248  White\_clover | 768  (22 rows) |

* Only a single term “Deer” is present

1. Check what enzyme nomenclature is present

|  |
| --- |
| agrbrdf=> select enzyme,count(\*) from gbskeyfilefact group by enzyme order by 1;  enzyme | count  ------------+-------  ApeKI | 9053  ApeKI-MspI | 96  Pst1 | 288  Pst1&Msp1 | 96  PstI | 22758  PstI-MspI | 288  PstI/MspI | 96  (7 rows)  agrbrdf=> |

* We will include both “Pst1” and “PstI”

1. From 1 and 2 design the query. It is probably best to “lower-case” the species and enzyme terms as shown to be sure that case variation doesn’t affect the query. Also included in this query is some standardisation of the enzyme name, just for illustration (not really necessary as Tassel does not AFAIK use the enzyme column of the key-file). (Note there are variations such as Pst1 that are not covered by the example code below, but these would be easy to handle).

|  |
| --- |
| Select  Flowcell,  Lane,  Barcode,  Sample,  PlateName,  PlateRow as Row,  PlateColumn as Column,  LibraryPrepID,  Counter,  Comment,  regexp\_replace(regexp\_replace(enzyme,'[/&]','-'),'ApeKI-MspI','MspI-ApeKI','i') as Enzyme,  Species,  NumberOfBarcodes,  Bifo,  control,  Fastq\_link  /\*, Sample||':'||LibraryPrepID as FullSampleName #uncomment this line for Tassel5 \*/  from  gbsKeyFileFact  where  lower(species) = 'deer' and  lower(enzyme) in ('psti', 'pst1')  order by  flowcell,  lane,  barcode; |

1. Execute the query from the psql command-line , writing out data in required format

|  |
| --- |
| The query is as above.  Before executing the query in psql, enter the following commands – this will  Send tab-delimited output to a file mykeyfile.txt  \a  \f '\t'  \o mykeyfile.txt  \pset footer off |

# References

[Construction of relatedness matrices using genotyping-by-sequencing data](http://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-015-2252-3), Ken G. Dodds, John C. McEwan, Rudiger Brauning, Rayna M. Anderson, Tracey C. van Stijn, Theodor Kristjánsson and Shannon M. Clarke, BMC Genomics201516:1047

# Appendices \*\*\*\* Please Ignore This is Under DEVELOPMENT! \*\*\*\*\*

## Lab DataFlow

### Data Flow Overview

The starting point for the data flow into the AgResearch HiSeq lab is a spreadsheet describing the sample in the case of plant data, and direct entry of samples into the Genophyle system in the case of animal data. Data supplied in spreadsheet form for plant samples is manually transcribed into Genophyle.

The initial record in Genophyl is a “Stock Plate”. This is basically the initial plate that sample is plated out onto. The stock plate is not itself used for subsequent quantitation steps. GBS stock plates are identified via a prefix in the description – for example *GBS\_SQ0152\_Stofn IPN Progeny Plate 13\_PstI\_F1/R1*. The database table involved is *t\_Plate\_Stock*.

The initial quantitation is for DNA concentration, which is done using the Nanodrop2 machine. The sample is re-arrayed to what is known in Genophyle as a “PCR plate”. This step is managed in the database by the tables *t\_Plate\_PCR* and *t\_stockPlatePCR*. For example the PCR plate corresponding to the above stock plate can be obtained as:

|  |
| --- |
| SELECT p.description from  (t\_Plate\_PCR as p join t\_StockPlatePCRPlate as link on  link.PCRPlateKey = p.PlateKey) join t\_plate\_stock as s on  s.PlateKey = link.StockPlateKey  where  s.Description = 'GBS\_SQ0152\_Stofn IPN Progeny Plate 13\_PstI\_F1/R1' |

* i.e. *(1841) 96w GBS\_SQ0152\_Stofn IPN Progeny Plate 13\_PstI\_F1/R1*

Key-file creation is currently done via a convoluted process which starts from a PCR file extract.

### Source and intermediate data-file examples

|  |  |
| --- | --- |
| Data file description | Link to example |
| Spreadsheet describing a plant sample |  |
| Genophyle import CSV derived from original plant spreadsheet |  |
| Extract from database used to control Nanodrop2 (DNA concentration assay) |  |
| Extract from database used to |  |

### Screen Shots of Key Genophyle Interactions

|  |  |
| --- | --- |
| Step description | Link to screen-shot |
|  |  |
|  |  |
|  |  |
|  |  |

### Key-file Extract Stored Procedure Development

A database stored procedure is currently being developed for extracting keyfiles. The skeleton is as follows:

|  |
| --- |
| --  -- this is a work-in progress as at 11/2015. It will extract gbs keyfiles. It takes 3 arguments  -- as described , which control the content and format of the extract  --  CREATE FUNCTION [dbo].[f\_gbs\_extractKeyFile]  (  @sqnumber\_listname varchar(64), -- a list of SQ numbers that this keyfile should contain  @format varchar(64), -- format of extract. Options are T3, T5  @stage int -- stage 1 = no flowcell id yet. stage 2 = with flowcell id    )  RETURNS  @key\_file TABLE  (  flowcell varchar(32),  lane int,  barcode varchar(32),  sample varchar(32),  platename varchar(32),  platerow varchar(32),  platecolumn int,  libraryprepid int,  counter int,  comment varchar(256),  enzyme varchar(32),  species varchar(256),  numberofbarcodes int,  bifo varchar(256),  fastq\_link varchar(256),  resultid int  )  AS  BEGIN  insert into @key\_file(flowcell)  values('something')  -- etc  return  END |