TheSNPpit
User Guide
A high performance database system for managing large scale SNP data

Eildert Groeneveld
Institute of Farm Animal Genetics
Friedrich Loeffler Institute
Federal Research Institute for Animal Health
Mariensee
D-31535 Neustadt
Germany

May 29, 2019

Contents

TheSNPpit is a database system for managing large scale SNP genotype data in a highly processing in the order of 70mio SNP genotypes per second. Through its novel subset definition system it allows the definition of subsets of original data at basically no storage cost. Subsets can then be exported by only specifying the subset names. It can handle all current genotype data from any panel size be they 1000, the 50K chips or the high density chips like the 700K cattle chip. Larger chips are no problem, with a panel size of 20mio having been smoothly handled.

TheSNPpit has implemented three new ideas: highly compressed vector storage, set based manipulation, and very fast retrieval which is written in C with Perl as the base for the framework. Data are stored in PostgreSQL. As an example, database size for the storage of 121921 animals with 3K, 13K, 54K, 215K, and 752K panels with a total of 22 billion genotypes amounted to some 16GB. Similarly, genotypes from 2045 animals with 22mio SNPs each totaling more that 40 billion genotypes amounted to around 15GB. The maximum tested thus far stored SNPs from various panel sizes on more than 18 mio samples with a total of 3.4 trillion SNPs in 860GB.

A completely generalized procedure allows storage of phenotypes, which will automatically be exported with the SNP genotypes for further down-stream processing.

Exports can easily be integrated in pipe lines as may be required for genetic evaluation in animal breeding.

1 Introduction and Objectives

High throughput single nucleotide polymorphism (SNP) genotyping is evolving at a staggering rate developing into a powerful tool in genetic analyses in all areas of biology. While its promises are immense so are the data processing issues associated with it. Dropping genotyping costs and ever increasing marker densities result in a huge increase in data volume, which seems to develop faster than the already impressive rate at which data storage costs have come down in the past. Thus, increasing data storage requirements along with an increase in processing time are an issue.

A number of developments can be observed which will lead to increasing data management problems.

Firstly, there is the increase in panel sizes. Micro satellites where quickly replaced by low density SNP panels starting with 5000 genotypes which then were rapidly replaced by 30000 (or 30K) panels and then increasing to 50 or 60K versions. As of this writing so called high density panels are in use with 700K and up to 4000K genotypes. This means, that the amount of data for an individual has increased by a factor of 1000. This is not a major problem in itself, but rather when looking at the file sizes involved as they are delivered from the genotyping labs. To give just a few examples: a data file comprising 90 samples genotyped with a 228K panel has a size of 80MB using the PLINK[5] data format. Using a 4000K panel on 90 individuals will result in a 1.3 GB in PLINK format.

A dataset from 404 samples genotyped with a 36K chip has the size of 1.5GB for a certain Illumina export format.

Secondly, many well developed software packages like PLINK and Genabel are freely available for data analysis. This areas is under constant development with new procedures and algorithms being developed and made available in rapid succession. All these packages use genotyping data in some predefined format, some of which have even developed into de facto data exchange format (like those used in PLINK). It is generally left to the users to supply and thus also manage the input data.

These packages are often used to define subsets of data going through a series of quality control steps, during which the data sets get reduced as more constraints (like major allele frequency (MAF) or Hardy-Weinberg-equilibrium (HWE)) are enforced. The resultant data set is very similar to the original and still of about the same size, thereby proliferating the number of datasets to be dealt with.

Thirdly, along with increasing panel sizes the number of individuals genotyped is increasing rapidly. In animal breeding the inclusion of SNP data has taken the genetic evaluation of breeding populations by storm. In the USA, each month new SNP data from 6000 cattle are added to the pool with regularly increasing panel sizes. These data have to be managed together and accessed by the individual's ID. Apart from increased – and drastically increased – data volume joint usage of panels of different sizes on the same or related individuals becomes an issue.

In genetic evaluation in animal breeding current and historic animals are included in one joint evaluation. Therefore, panels of different sizes need to be managed. This can either be done by using a common set of SNPs across different panels or imputing higher density genotypes from the lower density panels, which may obviously increase data volume dramatically. Here an efficient management of different genotype sets becomes obvious.

The scale of the problem can be demonstrated by a few examples. A few years ago micro satellites where used as genetic markers. Per individual tens of markers were generated and used. This resulted in small datasets of perhaps 50 markers or variables per individual. Large dataset would perhaps comprise a few 1000 individuals, resulting in something like 50000 data elements. With the emergence of the first SNP panels of a few 1000 the total number for the same number of individuals would go up to 5000000. In rapid succession the panel size went from 30000 (30K) to 60K to 700K to a few million, pushing the 50000 data elements to 2000 mio genotypes for the hypothetical sample of 1000 individuals genotyped with a 2000K panel. To obtain the order of magnitude of storage requirement we assume 1 byte per genotype and arrive at something like 2 terrabyte storage. Disks of this size are available but our example only comprises the relatively small number of 1000 individuals. Thus, increasingly, SNP data management is getting more attention.

Currently, most users of SNP genotype data employ files for basic data storage. Its advantage lies in the fact, that use of downstream data analysis software is straight forward, as packages like PLINK provide import facilities of certain data formats as supplied by the genotyping labs. Thus, with new data in a new project, a relatively easy start is possible. However, with increasing data volume – as indicated above –

the management of what was a handy 30K dataset becomes very cumbersome as the development continues and genotyping record sizes increase. It is to be expected, that this will become very apparent in the near future for many labs.

It is generally accepted knowledge that files are not well suited to the management of structured data with many entities where the attributes need to be stored in the file names. Here databases are much more appropriate and have largely replaced data storage in files. The same reasoning also applies to the problems described here. Accordingly, relational database management system (RDBMS) should also be used for genotype data.

Apart from the sheer management problem of many large datasets, the issue of increased data processing time even in the data preparation phase becomes a directly linked issue: while processing small files is never a computational hurdle, terrabytes of data volume will result in increasingly unmanageable computing times. The same observation can be made when data is stored in an RDBMS unless ways are found to address the data volume and data management issues. Adressing these issues lead to the development of TheSNPpit. For a full discussion see Groeneveld & Lichtenberg [2].

1.1 Three new ideas

Three new ideas have been developed to address the three critical areas storage space, operational speed and management of SNP data. Benchmarking TheSNPpit against a few other database systems has shown that the new design is far superior both in terms of storage and export speed [2].

1.1.1 Compressed storage in database

Typical files from genotyping labs easily have sizes of hundreds of M byte per individual. Their structure is determined through the genotyping panel, which is characterized by the number of SNPs involved and their name along with their positions in the genome. We propose to store the SNP genotypes of one individual in a compressed vector (*Genotype_vec*) using the position as determined by the panel map, which leads to one genotype record per individual. When only the biallelic state of a SNP is of interest, two bit storage is sufficient, allowing 16 SNPs to be stored in one 32bit integer word. Accordingly, all genotypes from a 60000 SNP panel can be stored in an integer vector of dimension 3750 or 14.6 kb. Once a panel map is be stored, any number of resulting genotypes can be loaded.

1.1.2 Set based manipulation

For easy data manipulation, we introduce the concept of effectively spaceless genotype sets. Each SNP panel comprises a specified set of SNPs. Using the SNP's position in the panel as its position in the genotype bit vector enables access to each SNP without explicitly having to state the SNP name. Once genotypes are treated as vectors, a SNP dataset can be viewed as a matrix of genotypes with the SNPs constituting the columns while each genotyped individual leads to one row.

Often, only subsets of SNPs are used in analyses, perhaps only those from a particular chromosome, or SNPs with a minimum frequency. Using a bit vector *SNP_sel_vec* of the dimension of the SNP panel provides a generalized approach: a .TRUE. or .FALSE. decides if a SNP is a member of a particular subset. A genotype set is then completely defined by adding a vector *Indiv_sel_vec* which contains the individuals of the subset. Finally, a set name for the combination of a particular *SNP_sel_vec* and *Indiv_sel_vec*, uniquely specifies a genotype set. Thus, creating new derived genotype sets amounts to creating two new lists: one for the SNPs and the other for the individuals and giving this a genotype set name for easy access. The storage implications are clear: instead of having to store a matrix of dimension $n_{snp} * n_{indiv}$ only two vectors of dimension n_{snp} and n_{indiv} need to be stored for each derived matrix, which are thus effectively spaceless.

1.1.3 Fast data retrieval

Being a central repository for all data to be analyzed, fast exports are critical. A data analysis step typically starts with an export using an appropriate format for downstream processing. Often exports are performed through costly SQL selects on the basis of SNP names [1, 4, 6]. However, export speed is a function of both the data storage and the retrieval scheme. Storage overhead of our approach is minimal: the *Indiv_sel_vec* is an integer vector with as many 32bit words as there are individuals in the genotype sample, while *SNP_sel_vec* has the dimension of the number of SNPs. Thus, all genotypes from a 1 mio panel will occupy less than 1MBbin *SNP_sel_vec*. Clearly, on this data volume basis our design will have a performance advantage over the OSPR paradigm which needs to process 1 mio records for each individual.

For data retrieval, the genotype set approach replaces SQL based SNP selection by much faster vector operation. An export of a genotype set amounts to these actions: firstly, *SNP_sel_vec* and *Indiv_sel_vec* are fetched for the chosen set. Secondly, for each individual the compressed SNP *Genotype_vec* is retrieved through one SQL select and shrunk on the basis of the *SNP_sel_vec* which are implemented as fast shifts [2]. Thus, the extraction speed is largely independent from the subset selection.

The benchmarking results from the proof of concept implementation were already an order of magnitude faster than the export times found at that time in the literature[3]. However, it was felt that the execution speed should be able to get increased beyond that limit. Time critical parts of the data transfer from the database to the user space and the unpacking and recording was rewritten in C. This resulted into a further speed improvement by a factor of 100. As a result, a 12GB ped file with 5500 samples on a 580K panel exports on a four year old laptop in 68sec wall clock time for the 3.2 billion SNPs. With this speed, frequent exports for down stream analyses are a snap, allowing to delete such export file without much thought as they can be re-exported very quickly.

The above ideas have explained and presented in [2] along with extensive benchmarks.

2 The Implementation

TheSNPpit[1] has been written in Perl with compute intensive parts in C. It uses PostgreSQL as a database backend and can run on any up to date Linux system.

2.1 Naming conventions

Many examples will be presented to clarify issues. They are contained in boxes with a black frame. Sometimes the lines are too long to fit the box. If that happens, less important parts (eg time stamps) are shortend with '...' indicating removed parts.

As has been outlined above, genotype sets are at the core of TheSNPpit. Each of them is defined by only one SNP selection vector and one individual selection vector. Thus, interaction with the database is based on these three elements. TheSNPpit uses a fixed format for these element: they have the names *gs_nnn*, *is_nnn*, and *ss_nnn* denoting a genotype set, an individual selection vector and a SNP selection vector with nnn being a three digit number.

Upon importing data the names are automatically created starting with *gs_001*, *is_001* and *ss_001*. Listing 1 gives an example of a report on genotype sets. Along with the genotype set name, here going from *gs_001* to *gs_006*, we have the individual selection vectors and the SNP selection vector. *gs_001* originated from the first import of data into the empty database. Accordingly, all three names start with 001. The *is_001* will contain all sample IDs that were loaded from the file *14chick-20.ped*, while the *ss_001* selection vector will have all SNPs in the panel activated. *gs_002* is the result of adding the content of file *14chick-add1.ped* to the existing panel with data from *14chick-20.ped*. Not surprisingly, *gs_002* is based on the SNP selection vector *ss_001*, as here also all SNPs get imported. In contrast, a new individual selection vector is created: the *is_002*, which contains only the sample IDs from the current load. However, usually new data is added to the already existing SNP. For this reason another genotype set is created, the gs_003. Its individual selection vector, *is_003*, now comprised sample IDs from the initial load plus those from the current. Thus, if the user wants to analyze the current batch of data, she will export *gs_002*. If, on the other hand, she wants to look at all data imported thus far, *gs_003* will get exported.

In short, the principle for creating the three names is simple: the new one is an increment of the previous as illustrated in Listing 1.

[1] Eric Anderson was the first to use SNPPIT for his program for Parentage and Intergenerational Tagging for SNP `https://swfsc.noaa.gov/textblock.aspx?Division=FED&ParentMenuId=54&id=16021`. Eric came up with the idea to use TheSNPpit for this software: "I'm gonna toss my data into TheSNPpit". Grand idea! Thanks Eric!

Listing 1: Example of report on genotype sets, individual and SNP selection

```
1
2                              List of Genotype Sets
3
4  SetName|IndSel|SNPSel|Panel|   Timestamp |                    Comment
5  -------|------|------|-----|-------------|---------------------------------------
6  gs_001 |is_001|ss_001| 57K |2016-08-11..|[Full SNP sel. IDs only from pick.ped]
7  gs_002 |is_001|ss_002| 57K |2016-08-11..|MAF: 57636/38873 0.010
8  gs_003 |is_001|ss_003| 57K |2016-08-11..|MAF: 38873/38200 0.020
9  gs_004 |is_001|ss_004| 57K |2016-08-13..|MAF: 38200/37593 0.030
10 gs_005 |is_001|ss_005| 57K |2016-08-14..|MAF: 37593/36845 0.040
11 gs_006 |is_002|ss_006| 57K |2016-08-21..|NCind:48/46 0.020 NCsnp:36845/35623 0.03
12 gs_007 |is_003|ss_007| 57K |2016-08-21..|NCsnp:36845/35623 0.03 NCind: 48/47 0.02
13
14                             List of individual selections
15
16 Panel|IndSel |nInd|Source |  Timestamp  |                Comment
17 -----|-------|----|-------|-------------|-----------------------------------
18  57K |is_001 |  48|Initial|2016-08-11..|[Samples from this APPEND]
19      |-is_002|  46|is_001 |2016-08-11..|NCind:48/46 0.020 NCsnp: 36845/35623 0.03
20      |-is_003|  47|is_001 |2016-08-11..|NCsnp:36845/35623 0.03 NCind: 48/47 0.02
21
22                               List of SNP selections
23
24 Panel|   SNPsel   |nSNP |Source|  Timestamp  |                 Comment
25 -----|------------|-----|------|-------------|-----------------------------------------
26  57K |-ss_001     |57636|  -   |2016-08-11..|[Full SNP selection]
27      |--ss_002    |38873|ss_001|2016-08-11..|MAF: 57636/38873 0.010
28      |---ss_003   |38200|ss_002|2016-08-11..|MAF: 38873/38200 0.020
29      |----ss_004  |37593|ss_003|2016-08-11..|MAF: 38200/37593 0.030
30      |-----ss_005 |36845|ss_004|2016-08-13..|MAF: 37593/36845 0.040
31      |------ss_006|35623|ss_005|2016-08-14..|NCind:48/46 0.020 NCsnp: 36845/35623
32      |------ss_007|35623|ss_005|2016-08-21..|NCsnp:36845/35623 0.03 NCind: 48/47
```

Listing 1 gives the lists for the SNP and individual selection that go with the genotype sets in line 6 through 12. Lines 7 through 12 give the subset definition (described further down) which are based mostly on the previous genotype sets. This is shown in the SNP selection vectors in lines 26 through 32. Up to *ss_006* each of them is derived from the previous SNP selection vector as shown in the column *SOURCE*. Only *ss_007* is based on *ss_005* which is the basis of *gs_005* as shown in line 10.

The observant user will notice that after 999 genotype sets this naming scheme will hit a brick wall. But even well before that limit is reached for either `gs`, `is`, and *ss* the, listings will become cumbersome to use. Further, many intermediate genotype sets will have become irrelevant and can thus be deleted.

Let us assume that, after checking the import in *gs_002* are considered not OK, then we could delete this genotype set. Now we can expand the naming scheme: for a new gs, is, and ss simply the first free number is chosen. After having deleted *gs_003*, the next new genotype set will again be called *gs_003*. If it is also an append action of new data, then also a new *is_nnn* is to be created, which will be *is_003* as the previous *is_003* was deleted with the *gs_003*. The effect of such deletes and reuse are shown in Listing 2.

Listing 2: Example of report on genotype sets, individual and SNP selection

```
eg@eno:~/database/snp/regression/tmp$snppit -R genotype_set

                                  List of Genotype Sets

SetName|IndSel|SNPSel|Panel|    Timestamp |                    Comment
-------|------|------|-----|------------|---------------------------------------
gs_001 |is_001|ss_001| 57K |2016-08-11..|[Full SNP sel. IDs only from pick.ped]
gs_002 |is_001|ss_002| 57K |2016-08-11..|MAF: 57636/38873 0.010
gs_004 |is_001|ss_004| 57K |2016-08-13..|MAF: 38200/37593 0.030
gs_005 |is_001|ss_005| 57K |2016-08-14..|MAF: 37593/36845 0.040
gs_006 |is_002|ss_006| 57K |2016-08-21..|NCind:48/46 0.020 NCsnp:36845/35623 0.03
gs_007 |is_003|ss_007| 57K |2016-08-21..|NCsnp:36845/35623 0.03 NCind: 48/47 0.02
gs_003 |is_004|ss_008| 57K |2016-08-22..|NCsnp:37593/36357 0.03 NCind: 48/47 0.02

eg@eno:~/database/snp/regression/tmp$snppit -R snp_selection

                                   List of SNP selections

Panel|   SNPsel    |nSNP |Source|  Timestanp  |                 Comment
-----|------------|-----|------|-------------------------------------------
 57K |-ss_001     |57636|  -   |2016-08-11..|[Full SNP selection]
     |--ss_002    |38873|ss_001|2016-08-11..|MAF: 57636/38873 0.010
     |---ss_003   |38200|ss_002|2016-08-11..|MAF: 38873/38200 0.020
     |----ss_004  |37593|ss_003|2016-08-11..|MAF: 38200/37593 0.030
     |-----ss_005 |36845|ss_004|2016-08-13..|MAF: 37593/36845 0.040
     |------ss_006|35623|ss_005|2016-08-14..|NCind:48/46 0.020 NCsnp: 36845/35623
     |------ss_007|35623|ss_005|2016-08-21..|NCsnp:36845/35623 0.03 NCind: 48/47
     |-----ss_008 |36357|ss_004|2016-08-22..|NCsnp:37593/36357 0.03 NCind: 48/47

eg@eno:~/database/snp/regression/tmp$snppit -R individual_selection

                                 List of individual selections

Panel|IndSel |nInd|Source |  Timestamp  |               Comment
-----|-------|----|-------|-------------|-----------------------------------
 57K |is_001 |  48|Initial|2016-08-11..|[Samples from this APPEND]
     |-is_002|  46|is_001 |2016-08-11..|NCind:48/46 0.020 NCsnp: 36845/35623 0.03
     |-is_003|  47|is_001 |2016-08-11..|NCsnp:36845/35623 0.03 NCind: 48/47 0.02
     |-is_004|  47|is_001 |2016-08-11..|NCsnp:37593/36357 0.03 NCind: 48/47 0.02
....
....
```

2.2 Supported import data formats

It is the objective to import directly into TheSNPpit the commonly used formats of called genotype data as they are delivered from the genotyping labs.

At this stage three import formats are supported. The first is the PLINK/ped format with AB genotypes. As a derivative, also other alleles like *ATGC* can be imported based on an expanded map file, further, any biallelic system can be imported. The second format is the *0125* format, which is widely used in animal breeding, coding not alleles but the three genotypes and the no calls. Should that not be the case, then the user will have to write a little script to reformat their data into either of the two forms.

The Interbull specified ASCII format *ib705* is very similar to the the *0125* format. It is specified as file format 705. Its major difference is that A and B codes are swapped: a SNPpit internal code that gets exported as AA with the 0125 format becomes a BB if the *ib705* format is used for export.

No standards exist for the import formats for phenotype measurement (except the rather reduced option of storing a few in the PLINK ped file. Here, we have implemented a general form that allows for handling any phenotype or possibly other data associated with a sample ID. For detailed description see: 15.

2.2.1 PLINK format

The PLINK map/ped files represent a very common data exchange format. It has the columns chromosome, SNP identifier, position in morgan, and basepair coordinate.

The default PLINK PED format denoted the two possible alleles are A and B, resulting in genotypes AA, AB, and BB. Together with the 0 for no call, this makes 4 states that can be accommodated in the two bits internal storage of TheSNPpit: AA becomes 0, AB and BA 1, and so on.

This biallelic system is actually a special case of any biallelic coding that can be handled, which does however require additional information in the map file. This requires some background information.

The coding system is defined on the panel level. Remember (or read up on the panel description), that content of the panel in terms of SNPs, their chromosome and other positions in stored in the table SNP of the database. Above this information (which is the contents of the map file) we have the field ALLELES, which is designed to store the two legal alleles for each SNP. The default is set to AB, i.e. if nothing is specified in column 5 of the map file, AB is assumed and inserted for each SNP. Each import for this panel will check for only A or B being used. For anything else the import will abort.

This opens the way for any other biallelic system: if you feel like coding your SNP alleles as I and J you need to add a fifths column to the map file that consists of 'IJ'. Obviously, your SNP data file is only allowed genotypes I and J (plus 0 for no call).

While this is a pretty useless execise it opens the way for something useful, i.e. the storage of ATCG format, preserving additional information. The required procedure should be clear: the map file needs to expanded by a column five. Contrary to the AB and IJ example, now it will not consist uniformly of AB or IJ for each SNP but rather the base pair combination that does indeed exist for it. The mapping onto the internal 2 bit vector follows the above description: the first character is mapped onto 0 for the homozygous, the second homozygous becomes 2, while the heterozygous becomes 1.

The behavious of export can be readily extrapolated from the above. On chosing the *-f ped* for export, the content of the field ALLELES in table SNP (which have created by the initial definition of the panel via the map file) is used for decoding the internal *0123* translating the 0 into the homozygous of the first character in the ALLELES field, the 1 into the two characters and so on.

In short, on export te PED format can be viewed as exporting the SNP presentation as it has been imported: if imported as AB it can never be exported as ATCG (however,

the opposite is possible, see the 0125 format).

Listing 3: PLINK map/ped file

```
hapmap.map
0 rs10399749 0 0
0 rs2949420 0 0
0 rs2949421 0 0
0 rs2691310 0 0
0 rs4030303 0 0
0 rs4030300 0 0
0 rs3855952 0 0
0 rs940550 0 0
0 rs13328714 0 0
0 rs11490937 0 0
hapmap.ped
#$fid,$id,$sid,$did,$sex, $pt, $gt_str
FAM001 NA18526 0 0   f    0    A B A A - - A A A A A A - - A A A A A A
FAM001 NA18524 0 0   m    0    A B A A - - A A A A A A - - A A A A A A
FAM001 NA18529 0 0   f    0    A B A A - - A A A A A A - - A A A A A A
```

2.2.2 0125 format

The *0125* format map/ped is given in Listing 4. The map file is similar to the PLINK map file, but has only the three columns *SNP name*, *Chromosome* and *base pair position*. The genotype file has one line for each sample, starting with the sample identification in the first column block. It is followed by the genotypes without intermediate spaces or tabs with as many entries as there are lines in the corresponding map file. In the example in Listing 4 we have 5 columns with genotypes for each sample in the file 'file.0125gen' with an equal number of lines in the map file. The codings are 0,1,2,5 standing for homozygous first allele, heterozygous, homozygous second allele, and no call.

Listing 4: 0125 map/genotype file

```
file.map
RS0 1 1
RS1 1 22
RS2 1 42
RS3 2 3
RS4 3 7

file.0125gen
V0000004 00552
V0000005 02520
V0000006 01520
V0000007 15525
```

2.2.3 ib705 format

As already mentioned above, the Interbull specified ASCII file format 705 is very similar to the the *0125* format. Its major difference is that A and B codes are swapped: a SNPpit internal code that gets exported as AA with the 0125 format becomes a BB if the *ib705* format is used for export. Furthermore, the original Interbull 705 specification includes codes for imputed genotypes with a codings 7,8, and 9. These codes are not used in TheSNPpit, as it provides a native infrastructure for handling imputed data 13.

Consistent with all other import formats, TheSNPpit also here requires a map file. Its format is identical to the *0125* format.

The *ib705* format map/ped is given in Listing 5. The map file is similar to the PLINK map file, but has only the three columns *SNP name*, *Chromosome* and *base pair position*. The genotype file has one line for each sample, starting with the sample identification in the first column block. It is followed by the genotypes without intermediate spaces or tabs with as many entries as there are lines in the corresponding map file. In the example in Listing 5 we have 5 columns with genotypes for each sample in the file '*file.ib705gen*' with an equal number of lines in the map file. The codings are 0,1,2,5 standing for homzygous BB, heterozygous AB, and homozygous AA, and no call. Notice, that the internal TheSNPpit coding is reversed for A and B. Thus, an AA exported with *0125* format will become a BB if exported with *ib705*.

Listing 5: ib705 map/genotype file

```
1  file.map
2  RS0 1 1
3  RS1 1 22
4  RS2 1 42
5  RS3 2 3
6  RS4 3 7
7
8  file.ib705gen
9  V0000004 22550
10 V0000005 22502
11 V0000006 21502
12 V0000007 15505
```

For more information see the import section.

2.3 Supported export data formats

Data exports are one of the main tasks of TheSNPpit. They constitute a defined set of SNPs and individuals ready for downstream data analysis. Based on TheSNPpit's genotype concept, they are easy and quick to define and very fast to export for use as input for target downstream software like PLINK, GenABLE or computation of genomic breeding values. Once processed after the export, they can be readily deleted, as they are equally easy to recreate, thereby drastically reducing data storage requirements and easing data managament.

Currently, three export formats are implemented. These are the two PLINK formats

Table 1: genotype coding

0125	ib705	ped	ped	ped	internal coding			
0	0	AA	xx	AA	00			
1	2	AB/BA	xy/yx	AT/TA	01			
2	1	BB	yy	AA	11			
5	5	NC	NC	NC	10			

ped and bed, and the *0125* format as shown in Listing6 and its close relative the *ib705* format. For a detailed account on coding see the above import section 2.2.

The PED format is the same as the one used for and described in detail under the import section. This format produces two text files files: the first is the *ped* file with the genotypes, the second has the extension *.map* which is a list of the SNPs in the same order as they are stored in the *ped* file. Both can be read by any editor, provided the files is not too large which is easily the case for the ped files. This has one line for each sample and requires 4 character for each SNP. With a few 100000 SNPs the file sizes easily amount to many gigabytes.

Listing 6: Export formats

```
1 snppit -E genotype_data --name=gs_023 -f ped|bed|0125|ib705 -o ctl_wk45
```

The default export is the ubiquitous PLINK *map/ped* format (Listing 3). Furthermore, the *0125* and *the ib705* format are supported, which are both commonly used in animal breeding, with the digits indicating the number of alleles (0,1,2) and the SNP and the missing status as 5 for the *0125* and a swapped A and B for the *ib705* format as described above. Apart from these three ASCII file formats also the much faster binary PLINK *bed* format is supported. For a detailed discussion see paragraph 4.7.

3 Internal coding, import and export formats

TheSNPpit stores genomic data in a compressed format 2 bits per genotype as already mentioned above. 2 bits allow four states, with biallelic systems we can accommodate AA,AB,BB, and missing. Internally, these are all coded in two bits as 00, 01, 10, and 11.

4 The actions

TheSNPpit is a command line tool which requires all actions to be specified as command line parameters. The general format of the command line calls of TheSNPpit is listed in the -h help:

```
eg@eno:~$ snppit -h
Usage:
```

```
    TheSNPpit -[I|C|D|R|E|U|A|S] <action> [Options]
Options:
     -h | -? | --help         short help
        | --manual            print manual page
     -v | --version           print version
     Actions:
     -I | --import  <what>    import [panel|data|pheno]
     -C | --create  <what>    create [snp_selection|individual_selection|genotype_set|
                                      full_snp_selection|genotype_set_nodups]
     -D | --delete  <what>    delete [panel|individual|snp_selection|individual_selection|
                                      genotype_set|phenotype|genotype_data]
     -R | --report  <what>    report [panel|genotype_set|individual_selection|snp_selection|
                                      duplicates|phenotype]
     -E | --export  <set>     export [genotype_set|individual_selection|snp_selection|phenotype]
     -U | --update  <what>    update [sampleid]
     -A | --append  <what>    append [panel]
     -S | --subset  <set>     subset [genotype_set]
     -T | --test              testing environment, mainly for internal use
     Auxiliary Parameters:
     -i | --infile      <file>    input file
     -o | --outfile     <file>    output file
     -f | --format      <format>  format of input [ped,0125|ib705] or output [ped,bed,0125,ib705]
     -p | --panel       <name>    panel name
     -s | --individual  <name>    individual_selection (maybe not needed any more)
     -m | --major                 use 'A' for major allele on PLINK Export (only ped format)
     -n | --snp         <name>    snp name (or snp_selection in some context)
     -g | --name        <name>    subsequent specification of name
     -c | --comment     <comment> comment
     -x | --exclude_from <name>   exclude SNPs or individuals/samples in the input file from
                                  ss_/is_nnn
     -q | --quiet                 reduce verbosity
     -t | --time                  some basic profiling of elapsed times displayed
     -d | --disk                  show disk storage
        | --ncind                 no call rate limit for Individuals
        | --ncsnp                 no call rate limit for SNPs
        | --first       <snp|ind> which constraints is computed first
        | --maf                   major allele frequency
        | --chromo                list of Chromosomes '1 4 W'
        | --nodups                only samples with genotype record
        | --skipdups              on import skip record if sample already exists
        | --skipheader            skip the first line in the input file
        | --not                   exclude those Chromosomes
        | --dryrun                only test if samples exist in import phenotypes & genotypes
        | --imputed               flag for imputed genotype data
        | --keep                  if imputed, keep old (imputed) genotype records. Default is
                                  overwrite.
eg@eno:~$
```

import This function loads data into the database. The two functions import a new panel, i.e. the SNP specifications or the SNP map and secondly the genotype data for a number of individuals relative to the previously loaded panel.

append add new genotype data in an append mode to an already existing panel in the database.

export as the name says, this functions exports what is called a genotype set from the database in one of the three defined export formats to be used in down stream statistical software like PLINK. Further, phenotypes, contents of individual and snp selection vectors can be exported.

create is used to define a genotype set on the basis of two vectores: the snp selection vector and the individual selection vector. Each of them constitutes a list of SNP names or individual IDs, respectively. The combination of them is then a named genotype set.

report gives a print out of various entities in the database, like panels, genotype sets and their constituting selection vectors.

subset on the basis of a genotype set defined in the database create a new one on the basis of major allele frequency, call rate and chromosome number.

delete delete records in the database

update update data in the database

4.1 Import (-I)

This function, invoked as *--import* or *-I*, imports either a new panel, a genotype (i.e. SNP) data sets or phenotypes.

The current focus is on the PLINK format, i.e. a map file for the description of the SNPs and a ped file with one line for each sample containing the genotype, i.e. all SNP from the panel described in the map file.

For this general PLINK format two versions are supported. Firstly, there is the *AB* format, meaning that the one homozygous genotype is coded as ' A A', the other as ' B B' and the heterozygous obviously as ' A B'. With this coding the alleles from 'ATGC' are lost. Sometimes knowing them is of interest, so this format is also supported.

Note that with any biallelic input format like AB and ATGC, heterozygotes like AB and BA are treated alike as one and the same heterozygote.

The second in TheSNPpit supported file type is the genotype *0125* format which is used in animal breeding. It is more compact than the *ped* format but does not provide allele information. The *ib705* format is treated operational identical as the *0125* format, only the meaning of codes 0 and 2 are swapped.

All file types require annotation information which is handled in the map files.

4.1.1 General comments on Panel

Before any genotype data can be loaded, the panel has to be inserted in the database. The table *SNP* contains the SNP name, its position in the compressed SNP vector, the chromosome name and the base pair position and the alleles. As described above, all SNPs are stored in a genotype vector. This means, that the SNP vector itself does not know which SNP is stored on a certain position. Here the table *SNP* comes into the picture with its central function to link the SNP names on import of SNP data in the genotype vector position. Each import requires two files: one which stores the the SNP data in a vector (as an example look at the lower part of Listing 8) and a map file (same Listing) which has the SNP name and the order in the map file reflecting the order in the SNP genotype file. Note, that two SNP genotype files can have a different order of

the SNP genotypes as long as this change of order is reflected in an equal change of order of the lines in the map file.

The function of the SNP table is now to translate the order of different map/SNP genotype files into always the correct order in the SNP genotype vector in the database, which is done on the basis of the SNP names. Obviously, the SNP names in the map file must correspond to those in the SNP table and no changes in SNP names between the imports are allowed. Should this happen, TheSNPpit will abort with an error message.

The columns *chromosome* and *base pair position* are optional as is *alleles*. About the former two, more information is provided further down. The columns "alleles" allows storage of two characters. Its default content is AB meaning that upon import of map data for each SNP only A and B are allowed as shown in Listing 8. However, also other biallelic alleles may be specified such as AT CG .

All map files are – at one point – derived from a 'build', i.e. the assembly of the genome. From this the chromosome and base positions are taken for the map files. Often chromosome/base positions are not known, but determined in later builds. This means, that the chromosome/base position in a map file, and accordingly also in the database will become outdated with time. Therefore, it makes sense to update the panel information if chromosome/base position is relevant to the user.

4.1.2 Panel and Genotype Data

To understand pitfalls the relationship between panel and genotype data needs to be understood. Each genotype record is stored as a sequence of bits for all SNPs for one sample. Their description is stored in the the map table

The danger of not being careful with the mapping of SNP names between *map* and *ped* file is obvious: if a map file with a different order of the SNPs relative to the positions in the *ped* file is used in a new batch of SNP data, the import as such will run without a glitch. However, the results will be wrong: the first batch of SNPs in the database will refer to different SNPs than the second. Imagine, that the first two SNPs were swapped in the *map* file used for importing the second batch. When both batches are exported, the output columns for a given SNP will be a mixture of different SNPs without you ever having a chance to notice. Thus, on import you need to perform these steps:

1. the SNP input file, let us assume it has the name *ctl770K-week-23.ped* .

2. determine the map that correctly described the *ped* file. This should always be done explicitly: either the *ped* file comes from the genotyping lab along with its *map* file. Alternatively, you may have the explicit information that the SNPs are in the same order as they were before, then the *map* file used in the previous import is the correct one. In either case the user needs to manually create the file *ctl770K-week-23.map*, which can either be done by copying the correct *map* file into the same directory as the *ped* file or by setting a link. For an import of SNP data only the SNP data file is specified, not the *map* file. This requires an explicit user action that helps avoiding mapping errors.

The file naming conventions are simple: The SNP file name must have an extension, which can be anything but not '*.map*' as this is reserved for the *map* file. These are legal SNP file name:

1. *ctl770K-week-23.ped* – meaningful for PLINK *ped* format
2. *ctl770K-week-23.0125* – meaningful for the *0125* SNP file format
3. *ctl770K-week-23.txt*
4. *ctl770K-week-23.ext*

The corresponding *map* files would be the same for all 4: *ctl770K-week-23.map*: thus, if you specify any of the above SNP input data file name a *map* file with the name *ctl770K-week-23.map* must exists in the same directory as the SNP data file. If it does, TheSNPpit will abort and tell you that the *map* file is missing.

Sometimes the first line in a *map* or genotype file is a comment which describes the the columns in that file. If not skipped, such a line will lead to a program abort. The *--skipheader* will do just that: ignore the first line and proceed from there. Beware, if *--skipheader* is used on an input file which does not actually have a comment line, the first line will of course get skipped nonetheless and you may be importing less data than you think. By the way, *--skipheader* can be used on any input file.

4.1.3 *AB* Genotype Data

Once the panel is loaded, data can be inserted into the database as shown in Listing 7.

Listing 7: Import SNP data

```
1 snppit --import data  -p panel-012 -i panel012.ped -f ped \
2                            [--skipdups] [--dryrun] [--skipheader]
```

The PLINK format is well defined, have a look at the PLINK documentation. Listing 8 gives an example. The first column in the map file is the chromosome which is handled as an ASCII string and the last is the chromosome position (i.e. *3*, *45497767*). The second column is the SNP name which is stored in *private_sid* in the database.

Listing 8: Example of the AB PLINK map/ped format

```
# the map file:
3 AX-81004456 0 45497767
11 AX-81004457 0 7144532
1 AX-81004458 0 72651292
4 AX-81004461 0 10677927
5 AX-81004464 0 27164428
13 AX-81004473 0 7778601
..
# the ped file:
1 84550 0 0 0 0 B B B B B B A A B B A A
1 84581 0 0 0 0 B B B B B B A A B B A A
1 84592 0 0 0 0 B B B B B B A A B B A A
1 84632 0 0 0 0 B B B B B B A A B B A A
1 85768 0 0 0 0 A A A A A A B B B B B B
1 85772 0 0 0 0 A A A A A A B B B B B B
1 85777 0 0 0 0 A A A A A A B B B B B B
1 85780 0 0 0 0 A A A A A A B B B B B B
1 85865 0 0 0 0 A A A A A A B B B B B B
```

The *ped* file i.e. the file with the SNP genotypes has one line per sample. For SNPpit the relevant information of the first six is the second column, which is the sample name, also used as an ASCII string. The genotypes follow from column 7 on with two letters per SNP, which have to be in the order as specified in the map file. Note, that for the default ped format *AB* coding is assumed.

The outcome of such a run is:

- the genotype data are loaded in the database
- the sample ID from the *ped* is used to create a record on the individual or sample and connect it to the genotype record. Should the sample name already exist in the database, this one is linked to the genotype record unless the flag *--skipdups* is set. In that case a message is printed stating the the sample is already in the database but that the genotype record is not inserted. Notice that this is also true if the particular sample has a genotype record in a different panel. To check for duplicates, by adding the option *--dryrun* to the import SNPpit invocation. Such a run will not attempt to import data but rather and only test for duplicate sample IDs.
- a SNP selection vector is create with all SNP activated
- an individual selection is created with the individual or sample ID taken from the current *ped* file
- a new genotype set is created on the basis of the snp and individual selection vectors.

If used repeatedly, for each import you will have one separate genotype set, which can for instance be exported or otherwise operated on (like maf etc., see the *-S* action). If, however, you intend to add today's SNP dataset to an already existing import in the

database, you would use the *--append* option which will create an individual selection vector containing all samples loaded thus far for the panel under consideration.

Each loading if SNP data requires, with the PLINK format, two files as indicated in the Listing 3. Thus, the map file is not only required, when creating a new panel in the database. Also, with each loading of SNP data, a map file with the same base name but with extension *map* has to exists in the same directory as the *ped* file. For example, if your genotypes are in PLINK format in file *weekp56K.ped* you also need to have the file *weekp56K.map*. As the names of each SNP genotype in the ped file needs to be known, so that they can be mapped each time into the same order in the SNP vector in the database, this order has to explicitly stated for each ped file in the corresponding *map* file.

This means, that the order of SNPs in the ped can be different from one week to the next, but this change needs to be reflected in the corresponding map file. As a result, you should never!!! use only one map file with different *ped* files, unless you are absolutely, positively sure, that the order of the SNPs in the *ped* file will always be the same. Just imagine the situation, where the order has changed. Then, for part of the data, the same SNP will be found in different positions, but you will never find out, only your results may not make much sense. This would be a real nightmare. And this can be avoided by always explicitly ensuring that each *ped* file has its correct map.

During the remapping of the SNP genotypes in the initial stage of loading them, also each SNP name from the MAP is tested for existing in the panel already loaded before. Should this not be the case, the loading will abort.

Correct use of the flag *--skipdups* requires some back ground information. The default operation when inserting a genotype records is the creation of a new record for the sample using the sample name or sample ID from the ped file. If a sample with that name already exists, then a message is given but the genotype records is attached to that sample. A sample with that name can be in the database for a number of reasons. Firstly, there could be a double use of the same sample name for two different individuals. This is an issue that has been described in paragraph 5.2, a situation that should really be avoided. Secondly, the sample name in the database and the one in the import file actually refer to the same individual. Then a genotype record could have been created for a different panel, a situation which is perfectly OK. Alternatively, this individual has been re-genotype. In that situation the user need to clarify the situation and rectify the problem.

4.1.4 *ATGC* genotype data

Most everything said above is true for both genotype formats. Also here with the *ATGC* format a *map* and *ped* file is required.

Listing 9: Example of the ATGC PLINK map/ped format

```
# the map file:
3 AX-81004456 0 45497767 GT
11 AX-81004457 0 7144532 CT
1 AX-81004458 0 72651292 AG
4 AX-81004461 0 10677927 GT
5 AX-81004464 0 27164428 GT
13 AX-81004473 0 7778601 CT
..
# the ped file:
1 84550 0 0 0 0 G T C C A G G T 0 0 C C
1 84581 0 0 0 0 T T C C A G G G T T T T
1 84592 0 0 0 0 G G C T A A G G T T C C
1 84632 0 0 0 0 G G C T G G T T G G C C
1 85768 0 0 0 0 G T T T A G T T G G T T
1 85772 0 0 0 0 T T C T A A G T G T C T
1 85777 0 0 0 0 G G C T A A G G T T C T
1 85780 0 0 0 0 G T C C G G G T G T 0 0
1 85865 0 0 0 0 T T 0 0 A A T T G G T T
```

As can be seen in Listing 9 the major difference is the additional column 5 which specifies which two alleles are defined for this respective SNP. Now it must be noted, that this column 5 is not a PLINK standard format. Instead, this is an extension, that is required by TheSNPpit to obtain the necessary information for each SNP. As there is no standard format that we are aware of, it is the user's responsible to add this column to the the standard PLINK map file.

Additionally, the ped file will now contain the genotypes as shown in Listing 9. Clearly, each SNP will have only genotypes from the two alleles defined in the *map* file.

It should be noted, that this format does not impact on the 2 bit compression in TheSNPpit, thus, disk space required for the $ATGC$ and AB format is identical. However, with $ATGC$ coding, on export exactly the same $ATCG$ alleles will be written to the output that was used in the import .

It must be noted that the order of the two alleles as specified in the map determines which of them becomes homozygous 1. For instance, if GC is specified then GG would be the first homozygous and CC the second. If these genotypes would be exported in *0125* format, then GG would become "0" and CC would come out as "2".

During import, the biallelic state is checked. Thus, if more that two alleles are present (plus possible no calls) in the import data set, this is reported and the processing is stopped. Notice, that for a AB in the map file both, AB and BA is allowed in the ped file, which are treated identically as being the heterozygote. Upon export to an AB format, all heterozygotes will be translated into AB even if the import was based on AB. These same rules apply to $ATGC$.

Furthermore, a statistic is given about the frequency of genotype and for $ATGC$ the alleles.

Listing 10: Example of output from ATGC import

```
          Some SNPs:
       SNP  0 : CC -  3
       SNP  0 : CT -  4
       SNP  0 : TT -  3

       SNP  1 : GG -  5
       SNP  1 : GT -  2
       SNP  1 : TT -  3
       ..
    All genotypes across SNPs
           00 14627
           AA 47552
           AG 24194
           AT 23675
           CC 23936
           CT 23909
           GG 47329
           GT 23587
           TT 71191
    All combination of genotypes across SNPs
         (no calls not listed)
         1       -  AA
         161     -  AA AG
         7062    -  AA AG GG
         152     -  AA AT
         6990    -  AA AT TT
             ..
         1       -  CT
         177     -  CT TT
         184     -  GG GT
         6940    -  GG GT TT
         164     -  GG TT
         173     -  GT TT
         2       -  TT

No biallelic errors: all SNPs passed test.
```

4.1.5 Other biallelic genotype data

If no alleles are specified in the *map* file, *AB* is assumed in the *ped* file. The respective column "*alleles*" in the database is filled with AB as a default when no fifth column exists in the map file. However, any other biallelic coding is possible. If for instance a *12* coding is used in the *ped* file, then the fifth column is required in the map file containing *12*. Likewise, any other biallelic coding can be used.

Further notice, that only one coding is allowed for one panel. This means that it has to be defined in the import of the panel. All SNP data imported thereafter must follow the coding specified when the panel was created.

4.1.6 The *0125* and *ib705* data format

The *0125* and the ib705 formats are compact genotype formats, meaning that for each SNP only its state as homozygous 1 ('*0*'), heterozygous ('*1*'), homozygous 2 ('*2*'), and no

Table 2: Compatibility of import and export formats

imp\exp	PED	0125	IB705	BED
PED (AB)	as import.(AB)	0125	0125 (not IB705)	0125
PED (ATGC)	as import.(ATGC)	0125	0125 (not IB705)	0125
PED (any biall.)	as imported	0125	0125 (not IB705)	0125
0125	AA AB BB	0125	0125 (not IB705)	0125
IB705	AA AB BB	0125	0125 (IB705)	0125 (IB705)

call (*'5'*) is stored, and (*'0'*) and (*'2'*) swapped as far as their meaning goes. Accordingly, it uses only one character per SNP, leaving out blanks between the genotypes.

Listing 11: Example of the '0125' and 'ib705' map/genotype format

```
# ib705 map file: rsname, chromosome, position
RS1 1 22
RS2 1 42
RS3 2 3
RS4 3 7
RS5 3 9
...
RS20 24 12115

The ib705 genotype data for 20 SNPs
V0000004 00552202011202102210
V0000005 02520202001521212222
V0000006 01520105555520112200
V0000007 15525050021112011000
```

Thus, where the PLINK ped format needs 4 characters per SNP, here only 1 character space is required, making the files much smaller and the processing on export faster, see Listing 11.

Notice, that the default alleles assumed for the *0125* format is *AB*. This implies, that also PLINK ped files with *AB* SNP can be loaded along with *0125* files.

In the 0125 format it is often irrelevant if A and B is related to a defined base pair: in genomic selection the results are identical if an AA is considered '(0)' or '(2)'. Sometimes the 'A' does have a defined and, thus, standardized meaning, as is for instance the case in the Illumina platform which is the basis for the *ib705* file format. For more information go to both Interbull and Illumina.

The following table 2gives an overview about the various import and export formats.

The first column denotes the various imports, the first three importing alleles, while the last two import genotypes in the 0125 format. If your objective is to get only genotypes out of SNPpit which you may want to use in genomic evaluation all imports can be

exported through through all: PED, 0125, ib705, and indeed the binary bed. However, if you want to recover the ATGC information you can of course only do this from those panels that got imported with a column 5 in the map file specifying the alleles available for each single SNP. This information would be exported using the PED and the ib705 format.

4.1.7 Imputed data

Imputed data do not really differ from SNP data as they are delivered from genotyping labs. The major conceptual difference is that the SNPs are derived from an original lab genotyping, which means that we have to handle for the same sample one original and at least one derived i.e. imputed set of SNPs.

Operationally, first the original lab SNPs will have to be stored in SNPpit. For a set of samples original SNP records will get exported and used as input for imputing software like *Fimpute*. This input can be from lower density panels like 56K, while other samples can come from high density 770K. After the imputation a new dataset is produced, which is has the dimension of the high density panel, i.e. all samples from the low and high density panels will be 770K. While the original genotypings originated from different panels, the resultant imputed samples can get stored together in one – in this case – 770K panel.

Technically, the imputed data could get imported into the 770K panel from which the HD samples get extracted for the imputation. As is clear from the above, this would imply having to import another (imputed) SNP records for those samples that were part of the imputation process. While it is possible to import more than one genotype record for one panel in SNPpit, it will present a problem during export: which of the two should get exported? or both?

TheSNPpit has implemented a strategy, that gracefully handles this situation by introducing a new panel of the same dimension and content as the original HD panel. Let us assume that the latter is named *ctl770K*. Then for the imputed data a new panel has to be created in TheSNPpit with e.g. the name *ctl770K_imp*. The SNP content of the two panels will be the same; actually, the same map file can be used for its creation.

Listing 12: create impute panel

```
1 snppit --import panel -p ctl770K     -i ctl770K.map
2 snppit --import data  -p ctl770K     -i ctlweek43.ped
3 snppit --import panel -p ctl770K_imp -i ctl770K.map --imputed
4 snppit --import data  -p ctl770K_imp -i ctlweek43_imp.ped --imputed
5 ..
6 snppit --import data  -p ctl770K_imp -i ctlweek43_imp.ped --imputed --keep
```

Listing 12 gives a sequence of how to set up the *--imputed* infrastructure. In line 1 an ordinary panel named *ctl770K* is created with data to be imported – as we have done it so far – as shown in line 2. If imputed data is to be handled by TheSNPpit a new panel has to be created as show in line 3. For ease of use we have shown the same base name *ctl770K* and appended *imp* to give *ctl770K_imp* to help the user. The actual name of

the impute panel is irrelevant, what is important in its creation is the *--imputed* option as shown in line 4. This flags the panel as one that allows only imputed genotypes to be imported as shown in line 5 of Listing 12. If an attempt is made to import data without the imputed flag, the process is aborted. In this way only explicitly flagged imputed data are loaded. A further effect of the flagging during data import as 'imputed' leads to an update of a particular sample genotype should it already have been imported before.

Thus, imputed SNP data can be imported with the standard *--import* action with one option added to the command line: *--imputed*. Should such an import be attempted for a panel that was not created with the *--imputed* flag the process will be aborted with an error message. This avoids inadvertent importing data for a non impute panel.

The relevance of the imputed flag becomes effective only during the second round of imputation. Let us assume, that the an expanded data set has been generated as input for imputation. Then the imputed SNP records will consist of samples that do already have data imported to the panel *ctl770K_imp*, and some new samples that do not have an entry in *ctl770K_imp*. On this second import for panel *ctl770K_imp* will lead to those samples already imported before to be updated, thus overwriting the previous imputation. In a series of imputations, the most recent imputed SNPs will likely be considered 'better' i.e. more accurate than the previous. If this view is taken, it makes sense to overwrite the previous SNP record with the newly imputed version. Then, in the panel *ctl770K_imp* we shall always have only one record per sample. This means, that on export always only one record per sample will be retrieved. If used for genomic selection only the latest and 'best' SNP record will be extracted.

This procedure is exactly, what is implemented as a default in TheSNPpit. In short, whenever a record is to be imported with the *--imputed* flag, the database is checked if a SNP record is already stored. If that is the case, then this record is overwritten by the new, else a new record is inserted in the database. It is at this stage where the *--imputed* flag comes into the picture: the update will only be done to records with the *--imputed* flag set. This avoids the inadvertant overwriting of of SNP data, should the wrong panel e.g. *ctl770K* be specified.

Listing 13: Import SNP data

```
1 snppit --import data  -p panel-012_imputed -i panel012.ped -f ped --imputed \
2                       [--skipheader] [--dryrun] [--keep]
```

Listing 13 shows the complete syntax of the *--imputed* flag for data import. Neither the content of the input file nor its format is any different from the non imputed data import handling, except for the *--imputed* flag. Clearly, the panel *panel-012_imputed* must have been imported before. Also this is nothing special: the number of SNPs has to be the same as in the SNP data set to be imported. Note, that also with this SNP data import a corresponding map file needs to be in the same directory as the data file. In this case it has to have the name *panel012.map* (Listing 13). Clearly, the SNP names have to be those that have been loaded at map import.

As stated above, the imputed flag leads to a update of the genotype of already existing samples in that imputed-panel: in this way there are no duplicates and an export will only retrieve one record per sample. With a *--keep* flag added, no updating is done but instead a new record with the newly imputed genotypes is inserted. If this procedure is repeated, an increasing number of genotype records is created. Such a setup may make sense, if the change in imputed SNPs is to be analysed.

4.1.8 Proper Phenotype Data

While genotype data are reasonably well defined, a large variety of phenotypic measurements may be available on genotyped individuals. In TheSNPpit, phenotype storage is implemented using the EAV (Entity-Attribute-Value) model, i.e. a quite generic structure, allowing to store any number different traits or phenotypes along with individuals for which genotypes have already been stored. The added flexibility implies cautious action on the user side.

Listing 14: Import phenotypes

```
1 Sample_name,Breed,DG,FCE,BF,born,nba
2 14941_0,LR,566,2.34,12.1,13,12
3 262,LR,554,2.45,13.1,12,11
4 1552,LR,554,2.45,13.1,,9
5 1612,LR,766,3.2,14.2,16,8
6 2643,LW,776,2.11,12.4,14,12
7 66969_9,LW,888,2.55,,15,19
8 64749_9,LW,999,3.12,12.1,19,17
9 263_BKschg,LW,333,2.14,10.1,9,8
```

The starting point for import is a file which contains in the first column the sample name as it has already been used during the SNP upload. The following columns hold the phenotypes or traits. There is one header column which specifies the phenotype names. To avoid issues with capitalization all column names are converted to lower case upon input.

Listing 15: Import phenotypes

```
1 snppit --import pheno -i pheno.csv [--skipheader] [--dryrun]
```

In this way, any number phenotypes can be stored for an individual already in the database. Listing 14 gives an example of such a phenotype file. Their use is intended as follows: as you already know by now, TheSNPpit is a data repository with the ability to define subsets of data. These can then be exported for further analysis. To be as general as possible along with an export of a genotype set also a linked phenotype set will be exported. As an example, exporting a genotype set in PLINK format will produce a map file and a ped file. If phenotypes exist for animals in that genotype set, additionally, a third file, i.e. the phenotype file will be exported in csv format. This will contain in a

sort of spread sheet format all available phenotypes for the individuals in the genotype set. They can then base further action for analysis on this data set.

Care has to be taken when defining the header line, which effectively specifies the phenotype names which are used during later retrieval. Thus, specifying on import a phenotype as *daily-gain* and in another as *daily_ gain* will result in two different phenotypes, thus showing up on export in different columns in the csv files.

Further, the header names must not start will a number (like *2BF*) and should also not contain special characters like '+'.

The phenotype import has a so called *dryrun* feature. If this switch is set as shown in figure 15, only the sample names are tested against the database content providing a list of samples that are not in the database.

It is then up to the user to either fix the sample names or proceed with the loading by rerunning the command without the *--dryrun*.

Phenotypes are exported using the PostgreSQL crosstab functionality. The database presentation as EAV model (Entity-Atribute_Value) is translated into a cross table with the sample IDs as rows and the phenotype names as columns. This implies that only one line is produced for each sample listing all phenotype names as columns. If for a given sample the same phenotype name is used more than once with a phenotype value, only one will be shown in the exported csv file. Thus, repeated values for the same phenotype name for a given sample can be imported but will not get exported.

In short, only single and unique values - like breed, birthdate - can be handled effectively in this system .

An implication from the EAV storage mechanism is the fact that different traits for the same animal can be imported from different files. Thus, one file may have *ID,Breed,DG,FCE* while a second has *ID, born,nba*. Importing the second file will simply add the columns to the already existing in the database. Upon export a complete line is constructed with the header: *sample_ name,breed,dg,fce,born,nba*.

4.1.9 Other use for phenotype data

The phenotype infrastructure can be used for other objectives which are unrelated to typical phenotypic measurements. Internally, phenotypic data are stored in the database as strings. This means that you cannot do computations in the database on those phenotypic data that you may have loaded. But this is not what we want to do anyway. The objective of the phenotype solution implemented here is to be able to store additional data, for instance phenotypic data, along with the SNP genotype information. After an export of a genotype set, these phenotypes will be available in a *csv* ASCII file, that you can manage and process after export to your liking.

Apart from proper phenotypic measurements you could for instance store along with those the breed or the ecological niche that the sample belongs to. Then, after export, you could easily use a spreadsheet program to extract a list of samples belonging to a certain ecotype and use the list of resulting sample ID for a new subset definition in the database or for further processing.

One other use of the phenotype infrastructure is the storage of pedigree information or rather parental information, i.e. sire and dam. While the latter has been included in the *individual* table, loading sire and dam IDs are much easier to load through the pheno import action: once the SNP data are imported into TheSNPpit, the parents of these animals can be imported here. Note, that phenotype data can only be loaded for individuals already stored together with their genotype records. After export, the user takes over and picks up the pedigree data from the *.csv* file for whatever the purpose may be.

4.1.10 Import log file

When importing SNP data a log file is generated which holds the standard output to the console. Along with progress information on the loading process also information on duplicate sample ID are generated. The user may need to investigate such issues further and, therefore, does need this information in a file to which she can go back later on. The logfile name are of the form: *SNPpit-2016-05-12_21:40:11.log* . With a resolution of 1 second, each run will use a new log file.

4.2 Create (-C)

The create command is used to define subsets in the database. This is done either on the basis of files containing SNP or sample names or on the basis of already loaded sample or SNP selection vectors. This implies that no new SNP or panel data are loaded into the database. If external SNP names or sample ID have to be loaded they need to be available in an ASCII or text file, one line for each. These files are used as inputs in the CREATE actions.

While loading a panel and in particular SNP data potentially adds large volumes of data, making the database grow substantially, very little additional storage is required for by these create commands as this is only a list of sample IDs or SNP names. (Even if the latter would be million for a 1000K panel, the storage requirement is close to nothing compared to the SNP genotype data.)

The new (sub)sets are accessible through their symbolic name. SNP selection name have the format *ss_nnn*, individual selection vectors are identified as *is_nnn*, and genotype sets as *gs_nnn* . The naming is automatic: snppit starts with *ss_000* incrementing the nnn field by one each time a new set or selection is created. This behavior is only different when either *gs_nnn*, *is_nnn* or *ss_nnn* have been deleted. Then, the first free position is chosen. If, for instance, the *gs_011* has been deleted and *gs_001* up to *gs_010* exist in the database, then upon creation of a new genotype set the name *gs_011* will be chosen. The same applies to the SNP and individual selection vector names.

This naming mechanism is used whenever a new genotype set and individual and SNP selection vector is created, which is the case for *append* and *subset* action .

The general syntax is given in Listing 16.

Listing 16: General syntax of Create

```
snppit --create|-C [individual_selection|snp_selection|full_snp_selection|\
                    genotype_set]
   with auxilary parameters:
   -s  --individual_selection  file_name
   -x  --exclude_from          ss_name
   -c  --comment              'comments..'
```

4.2.1 Create snp selection

A genotype set to be used for exports is defined by a snp and an individual selection vector. Being able to define the latter two freely, will allow to define an export equally freely through linking any of the the two to form a new genotype set.

The command in 17 will create a new snp selection set using the set *ss_005* as a starting point and removing from it the 73 SNP in the file *excl-73SNP.txt* . The total number of SNP in the new set will then the number of SNPs in the *ss_005* set minus the 73 excluded in this run.

Listing 17: Syntax of create SNP selection

```
snppit -p chick500k -C snp_selection -x ss_005\
    -i 73SNP.txt -c 'exclude from ss_005 73 bad SNP'
```

Alternatively, a user might want to create a new subset with all SNPs to be included are in the snp data file. In that case the *--exclude_from* (or *-x*) clause is dropped and the calling is as given in table 18. This will create a new SNP selection vector and will get automatically a name like *is_056*. Such a situation may arise if in genomic selection a fixed set of SNPs is used that have been determined through research outside TheSNPpit.

Listing 18: Syntax of create SNP selection

```
snppit -p chick500k -C snp_selection  -i 430210SNP.txt -c 'GenSel. 430210 SNPs'
```

In the reporting section the new set will stand on its own and not be nested under a particular subset.

4.2.2 Create individual selection

As described above, a genotype set is defined through a snp selection and an individual selection. With the command in Listing 19 a new individual selection vector is created which – under a name like is_034 – creates a vector in the database containing the individual or sample ID for which genotypes must have been loaded before for the given panel. These IDs are read from the ASCII data file specified.

Listing 19: Syntax to create a new individual selection

```
snppit -p chick500k -C individual_selection -i 73Indivs.txt \
                       -c 'indiv.sel.from 73 Indivs'
```

Sometimes, it is useful to remove a few sample names from a particular individual selection. How this can easily be done is shown in 20. The difference to Listing 19 is that this is based on a defined individual selection, here *is_001*. The sample names to be excluded are taken from the file '*xclude2.txt*', one line for each sample.

Listing 20: Syntax to create new individual selection: remove a few

```
snppit -p chick500k -C individual_selection --exclude_from is_001 \
                                    -i xclude2.txt -c 'excl 2 individuals'
```

4.2.3 Create full snp selection

This action will create a new full snp selection vector for a given panel. With normal use, this action is not required, as such a full vector will be created when importing SNP data. However, the full SNP selection vector might have gotten deleted. In that case this action will simply create a new one, that can then be used to create a new genotype set by linking it to the appropriate individual selection.

Listing 21: Syntax to create a full SNP selection

```
snppit -p chick500k -C full_snp_selection -c 'recreate new full SS for chick500k'
```

4.2.4 Create genotype set

Combining a particular snp selection with a specific individual selection defines a genotype set with an automatically created name like *gs_123* 22. The next genotype set to be defined will then have the name *gs_124*.

Listing 22: Syntax of create a genotype set

```
snppit -C genotype_set -n ss_001 -s is_04 -c 'creates new GS'
```

Such a named genotype set is then ready for export.

4.3 Subset (-S)

Often, genotype sets are based on previous sets with the objective to further reduce this set or further clean it up. This can be a repeated process during which a number of new subsets will be created becoming consecutively smaller in terms of number of SNPs involved.

In this function a new subset is created solely on the basis of information in the database. Thus, no input files like for the *–create* action are required.

Clearly, all of this can also be done outside of TheSNPpit. One could even argue, that this feature goes beyond the original mission of storing and managing SNP data while leaving the processing to downstream software like PLINK. But then it became clear, that the features are not only very handy but exceedingly fast in terms of execution speed., Thus, we put is this feature.

The following possibilities exist for genotype subsetting:

1. the new subset is based on the no calls in the genotypes. Here, either SNPs can get removed that fall below a threshold *--ncsnp* or/and samples that fall below a threshold *--ncind*. Furthermore, it can be decided if the the SNP or the sample section should be done first, which may yield different results -- if unclear: think about it. As an example: when *--ncsnp=.10*, then all thoses SNPs are excluded from the new set that have more or exactly 10% no call. Equally, if *--ncind=.15* then those samples will get thrown out that have more or equal than 15% in all of their SNPs.

2. the new subset is based on a limit on major allele frequency(*--maf*).

3. the new subset is based on chromosome information provided the chromosomes have beein specified during panel import.

The available options are given in Listing 23. The first line creates a new genotype set based on gs_031 excluding all those SNPs with a no-call rate more or equal 3\%. This first step is indicated by the *--first = snp* flag. In a second scan, the no call rate for each sample is computed, removing those samples with a no call rate larger or equal than 5%. As a result a new genotype set is created base on a new snp selection vector and a new individual selection vector.

The second line also creates a new genotype set again based on both a new snp and a new individual selection vector. Only, this time, first a new individual selection vector is created and the snp selection vector.

In the third line a new genotype set is created only on the basis of a new snp selection vector by setting the no-call threshold to 3% and using the same individual selection vector as in *gs_ 031*.

In the fourth line we have a similar setup: here the SNP selection vector from *gs_ 031* is kept, while the individual selection vector is recomposed on the basis of the 5% no-call constraint.

Finally, the fifth line creates a new genotype set on the basis of a major allele frequency constraint of 4.6% resulting in a new snp selection vector as in line 3.

Listing 23: Syntax to create a new genotype set based on No Calls and major allele frequency

```
snppit -S genotype_set --name=gs_031 --ncsnp=.03 --ncind=.05 --first='snp'
snppit -S genotype_set --name=gs_031 --ncsnp=.03 --ncind=.05 --first='ind'
snppit -S genotype_set --name=gs_031 --ncsnp=.03
snppit -S genotype_set --name=gs_031 --ncind=.05
snppit -S genotype_set --name=gs_031 --maf=.046
snppit -S genotype_set --name=gs_031 --chromo='1 2 12 X' --not
snppit -S genotype_set --name=gs_031 --nodups
```

The second last line finally creates a new subset based on the gs_031 on the basis of the chromosomes: *--chromo='1 2 12 X'*. In this example all SNPs are selected which are NOT (because of the '-not') on chromosomes 1, 2, 12 and X. Notice, that the chromosomes are handled as character strings. Thus, they need to be specified as used in the database. If the '-not' is dropped, then the new genotype set would only contain SNPs from the four chromosomes 1, 2, 12, and X.

Sometimes, you may have duplicates in the database, that you do not want to have included in an individual selection vector or genotype set. Duplicates mean: for a given sample name there are more than 1 genotype vectors for a given panel. This action *--nodups* removes both or all duplicates from the sample ID list, as it cannot be decided which is the 'good' one. While this is done internally via a new individual selection vector, i.e. a new *is_nnn*, the process starts from a genotype set as shown in Listing 24.

Listing 24: Syntax to create a genotype set without duplicates

```
snppit -S genotype_set --name=gs_012 --nodups -c 'duplicates removed from gs_012'
```

Please note that such a new genotype set can also be created manually. The steps would get:

1. get a list of individuals: *snppit -R individual --name=is_012 > is_012.exp*
2. get a list of duplicates: *snppit -R duplicates --name=gs_012> dup_gs_012.exp*
3. remove the ID with duplicate genotypes from file *dup_gs_012.exp* in file *is_012.exp*
4. import the reduced IS list: *snppit -p chick500k -C individual_selection -i is_012.exp*
5. link this new vector (which may have been reported as *is_013*) with the ss from *gs_008* (which may be *ss_007*): *snppit -C genotype_set -n ss_007 -s is_014 -c 'no dups'*
6. you can check if the new gs is indeed free of duplicates by running: *snppit -R duplicates --name gs_013*

4.4 Append (-A)

The accumulation of genotype data is usually a continuous process: as more samples are genotyped new datasets are received from the genotyping lab and must, thus, be loaded into the database. Listing 25 shows the syntax giving in line 2 an example. The logic behind the syntax is, that new genotypes are alway added to a given panel. This make the command and the information required very simple. Once the panel is specified, everything else is also implicitly defined.

This action creates two genotype sets. The first is based on the samples loaded in this run, while the second creates an individual selection vector based on all samples in the panel. The logic behind the two genotype sets is, that it documents which samples are loaded together. This is also reflected by the time stamps added to comment field as shown in Listing 26. The *is_097* holds those 200 sample IDs that are from the file *0009999-00200.ped*, while the is_098 comprises all samples loaded for panel *P.00009K*, i.e. 1500.

Listing 25: Syntax of Append

```
1 snppit --append panel --panel=<name> -i filename -f ped
2 snppit -A panel -p 770K -f ped -i week23.ped
```

The SNP selection list to be used in the new genotypes set will be the same as in the base genotype set: thus, no new selection vector has to be created. Only if for one reason or another (deleted by the user) no complete snp selection vector exists, a new one will be created. Clearly, new data can only be added to a conforming panel. Thus, trying to add new data with 500000 SNP to 65000 SNP panel will abort.

Listing 26: List of genotype sets and individual selections from Append run

```
1  snppit -A panel -p P.00009K -f ped -i 0009999-00200.ped
2
3  gs_085 |is_097|ss_008|2014-09-27 11:19:31| IDs only from 0009999-00200.ped
4  gs_086 |is_098|ss_008|2014-09-27 11:19:31| All IDs from 0009999-00200.ped
5
6  is_097| 200|Initial|2014-09-27 11:19:31| Samples from this APPEND
7  is_098|1500|Initial|2014-09-27 11:19:31| All samples in panel
8
9  P.00009K|-ss_008 |   9999|  -   |2014-09-27..|Full SNP selection
10         |--ss_009|   1002|ss_008|2014-09-29..|Chroms: 18 21 23 with 1002 SNP
```

The most common procedure will likely be to chose the last genotype set which comprises all samples in the panel and also all SNPs. In this way the new genotype set created will reflect all samples and SNPs available in the database which can then be used for further filtering. This will likely make subsets created before the appending obsolete, so in many cases these can be deleted from the database.

Each append requires the explicit definition of the SNPs in the genotype data file, i.e. the .ped file. Accordingly, TheSNPpit requires a map file with the same base name.

Thus, the genotype file 'week23.ped' will require a corresponding map file with the name 'week23.map'. It is the obligation of the user to ensure, that the ordering of th SNP in the ped file is reflected in the associated map file.

A --append run will result in the following:

- the genotype data are loaded in the database
- the sampleID from the ped is used to create a record on the individual and connect it to the genotype record. Should the sampleID already exist in the database, this one is linked to the genotype record
- a SNP selection vector is create with all SNP activated
- an individual selection is created with the individual ID taken from the current ped file
- a new genotype set is created on the basis of the snp and individual selection vectors.

Up to here, the actions are exactly the same as for the *--import* data under the *-I* action. Additionally, the append action also creates an individual selection vector which includes all sample IDs with genotype records in the current panel along with a new genotype set as stated above. This results in two genotype sets being produced by an *--append* run: the first contains only the samples currently loaded, while the second holds all individuals which have a genotype record for the given panel.

In this way the first genotype set documents each separate loading procedure, you have the date and the sample loaded. The sampleID can be exported through "*snppit -R individual_selection -s is_003*", which would allow targeted deletes if something went wrong. The second genotype set would be the one to use if you want to analyze the larger data set including the new data. Notice, that you may need to repeat the *--maf* or no call filtering based on this newly created 'full' genotype set.

4.5 Delete (-D)

This action allows to delete certain data in TheSNPpit database and must therefore be handled with utmost care. You can easily, by mistake, empty your database in this way. TheSNPpit has implemented what is called "cascading deletes". This means, if a record is deleted which is foreign key in other tables all the connected table content will be deleted as well. Deleting a panel name is a good example: all genotype data are linked to their respective panel. Thus, deleting panel name *Cattle700K* will also delete al genotype records from this panel, irrespective of this being 10 or 100000 records. This may sound scary – and it is. But not deleting attached data would leave them meaningless, once its panel had been deleted.

Make sure that you will survive, at least professionally, such a disaster by making sure that you have a backup you can go back to.

The database is a PostgreSQL database. Thus, it can be completely manipulated through psql, the PostgreSQL command line tool, which allows any SQL command to

be executed. Thus, if the user wants to perform an action on the database that is not supported by the TheSNPpit software, *psql* can be used. It goes without saying that the user needs to know what she is doing. Dropping all of the SNP data is a matter of typing "*drop table genotype_data*", and your 3TB of data are gone. You have been warned!

4.5.1 Delete a panel

This is a very strong delete which can, with one command, empty the better part of your database. Its syntax and an example is given in Listing 27.

Listing 27: Syntax of deleting a SNP panel

```
1
2 eg@eno:~/database/snp$ snppit -D panel --name P.00200K
3        CASCADing DELETE in Table: genotype_data 25000 records
4        CASCADing DELETE in Table: individual_selection 2 records
5        CASCADing DELETE in Table: snp 200000 records
6        CASCADing DELETE in Table: snp_selection 1 records
7  do you really want to go ahead? (deLete/N)
```

Along with the panel all genotype data are also deleted that are from this particular panel. Further, the associated snp and individual selection vectors will go and as a result of that also the genotype_sets that are made up of those selection vectors will also be delete.

This is specified in the *CASCADing* deletes output: from table '*genotype_data*' 25000 records will be deleted, 2 records from the individual_selection table and one from the *snp_selection* table. Further, all 200000 SNPs will be deleted from the *SNP* table. However, this will happen only if the user's response is indeed *deLete* with a capital '*L*'.

What is not deleted are the actual individual or sample IDs as they may be used in different panels. As a result you may have sample IDs in the database that do not have genotype data.

In a production database panels will likely not get deleted often if at all. Just like with any other delete: be careful, if you answer with *deLete* SNPpit take your word for it and not ask again.

4.5.2 Delete an individual

This deletes the sample or individual. Beware, that samples or individuals are accessed globally in the complete database. There is no subdivision within panels or breeds. Thus, if you use the same individual with two different panels this command will delete that animal and also through a cascaded delete the genotypes from both panels.

With a sample_ID also all connected genotype SNP data will be deleted as well as phenotypes that you may have stored. Furthermore, deleting a sample ID will impact on the individual selection vectors it is part of. Assume that sample *SAM4711* is to be deleted which is part of the individual selection vector *is_001* and *is_012*. Then, only deleting *SAM4711* from the *individual* table in the database would leave its two

references in *is_001* and *is_012* pointing to nowhere. Thus, TheSNPpit also removes *SAM4711* from both of them, thereby shrinking the two vectors by one.

The user should notice, that the genotype sets based on these two vectors *is_001* and *is_012* will also be different after the deletion: the export will have one record less than before.

Listing 28: Syntax of deleting a sample

```
1 eg@eno:~$ snppit -T -D individual --name=V0000004
2        CASCADing DELETE in Table: genotype_data 1 records
3        CASCADing DELETE in Table: phenotype 0 records
4  do you really want to go ahead? (deLete/N)
5 deLete
6 deLete you asked for it, now its too late and the delete goes ahead
7 DELETE FROM individual WHERE sample_id=1
8 eg@eno:~/database/snp/regression/tmp$
```

As with all other deletions the user needs to respond with the word *deLete*, notice the capital "*L*". Once typed and submitted the deletion process will go through and can not be stopped or reversed. In this example the sample name *V0000004* with the internal sample_id 1 was in fact deleted.

4.5.3 Delete a genotype set

This will delete simply this one record connecting an individual selection and a snp selection which forms the genotype set under discussion.

Listing 29: Syntax of deleting a genotype set

```
1 snppit --delete genotype_set --name gs_119
2 do you really want to go ahead? (deLete/N)
```

Thus, no real data get deleted. Using "*--create genotype_set*" can recreate this genotype set, as long as the individual selection and snp selection still exist.

4.5.4 Delete an individual selection

An individual selection is a named vector containing a number of sample IDs. With this command as shown in Listing 30 such an selection can be deleted.

Listing 30: Syntax of deleting an individual selection

```
1 snppit --delete individual_selection --name is_039
2 CASCADing DELETE in Table: genotype_set 1 records
3 do you really want to go ahead? (deLete/N)
```

Remember, that the individual selection vector together with the snp selection vector

defines a genotype set. Thus, deleting one of those will also render the genotype selection undefined and therefore useless. Thus, genotype sets which are based on a individual and also SNP selection vector will also get deleted. In this way you might lose access to your genotype data which will still be in the database.

You can gain access to the SNP data by creating through the import of individual selection a new individual selection vector and by linking this to an existing snp selection vector to create a new genotype set.

4.5.5 Delete a SNP selection

This will delete the snp_selection *ss_043* and through the cascading deletes all genotype sets which are based on *ss_043*.

Listing 31: Syntax of deleting an SNP selection

```
1 snppit --delete snp_selection --name ss_043
2 CASCADing DELETE in Table: genotype_set 1 records
3 do you really want to go ahead? (deLete/N)
```

The above paragraph on individual selection also applies here. Also here, you may loose access to your SNP data if you removed the complete snp selection created during initial loading of the SNP data.

4.5.6 Delete a phenotype record

This will delete **all** phenotypes stored in the database for sample *PIG011231*.

Listing 32: Syntax of deleting a phenotype

```
1 snppit --delete phenotype --name PIG011231
2 do you really want to go ahead? (deLete/N)
```

4.5.7 Delete genotype record

This action deletes one genotype record, i.e. all SNPs attached to a particular sample. A typical use is the case, when the same SNP record has been loaded more than once. In this case the command "*snppit -R duplicates --name=gs_001*" lists these as identical duplicates as shown in Listing 48.

Listing 33: Syntax of deleting a genotype record

```
1 snppit --delete genotype_data --name 22
2 do you really want to go ahead? (deLete/N)
```

So, if we wanted to delete the duplicate SNP records for sample *V0000005*, the command would be: "*snppit -D genotype_ data --name=*22". Notice that we need to specify here the unique genotype data ID which is different for the two records from the same sample. Running then "*snppit -R duplicates --name=gs_ 001*" will show two lines less: because we now have only one genotype record for sample *V0000005* it will not show up in the listing any more.

4.5.8 Cascading deletes

The following table 3 summarized the above deletions. The first column holds the deletion of one record of that particular type. The second column lists the associated cascading deletes which are connected to the record in the first column.

Table 3: Summary of cascading deletes

record deleted from	cascading deletes in
panel	genotype data, IS, SS, GS
individual selection (IS)	GS
snp selection (SS)	GS
genotype set (GS)	-
sample	phenotype, genotype data
genotype data	-
phenotype	-

4.6 Update (-U) sample ID

This function allows a batch update of sample IDs.

Listing 34: General syntax of updating sample IDs

```
1 snppit --update sampleid -i filename] [--dryrun]
2 snppit -U  sampleid --dryrun -i update-21.txt
3
4 head update-21.txt
5  NA18526, cattle_NA18526
6  NA18524, cattle_NA18524
7  NA18529,cattle_NA18529
8  NA18558, cattle_NA18558
```

As shown in listing 34, a sample ID can be easily replaced. The file contains the old sample ID and, separated by a comma, the new sample identification. It should be noted that leading and trailing blanks or white spaces will get eliminated.

For instance, you may have loaded genotypes with sample names that you need to change. Then you could export first those sample name that you need to update 35. This procedure can be used, if the sampleID used in the genotyping service is actually not the one that should be used in genetic evaluation. After the update, the genotypes

will only be accessible through the new sample names.

Listing 35: Export sample IDs to be updates

```
snppit --export individual_selection --name is_003 -o is_003.txt

head is_003.txt
 NA18526
 NA18524
 NA18529
 NA18558

vim is_003.txt
add new IDs

head is_003.txt
 NA18526, cattle_NA18526
 NA18524, cattle_NA18524
 NA18529,cattle_NA18529
 NA18558, cattle_NA18558

snppit -U sampleid -i is_003.txt
```

4.7 Export (-E)

As TheSNPpit is a data storage and data management package, it does not have its own data analysis capabilities. Instead, data sets have to be exported for down stream packages like PLINK. Therefore, exporting data sets is of prime importance. The general syntax is given in Table 36.

Listing 36: General syntax of the Exporting

```
snppit --export genotype_set|individual_selection|snp_selection|phenotype\
       --name=<what> [-f ped|bed|0125|ib705] [-o filename] [-m ]
snppit --export genotype_set --name=gs_041 --major
snppit -E snp_selection --name=ss_124
snppit -E individual_selection --name=is_213 -o sampleIDsfromis_213.txt
snppit -E phenotype --name gs_033 -o pheno-from-gs_003
```

4.7.1 Retrieving genotype sets

The minimum information required is the genotype set to be exported. If no output file name is specified a output file name will be created on the basis of the genotype set, the snp and individual selection name vectors. This results in a file name like: "*gs_003.ss_002.is_002.ped*" which is the genotype set *gs_003* which itself is defined through the snp selection vector *ss_002* and the individual selection vector *is_002*.

The default outputs (which can be changed in the file *TheSNPpit.conf*) are the PLINK map and ped files which can be directly further processed by PLINK. PLINK uses the coding '*A*' for the major allele. (Here it is not clear to me, what the actual relevance of

this is and whether it matters anywhere in the PLINK analyses). If the user wants to export data according to this convention, the genotypes will have to be scanned before export to determine for each SNP which of the alleles are in fact the major alleles in the current genotype set. Thus, all genotypes need to be processed twice: first to determine the major allele for each SNP and then, in the second pass to export them accordingly. This will obviously increase the processing time.

It should be noted that *--major* only makes sense if *AB* or *0125* is used in the input, either as a default or as column 5 in the map file. If it is specified for anything else, for instance with *ATGC*, then it gets switched off on export and a corresponding message is printed. Further, *--major* cannot be used together with the binary *bed* format and also does not make sense for the '0125' format.

The "0125" format exports the *AA*, *AB*, *BB*, and *NC* as *0, 1, 2*, and *5* in a dense format with no blanks between the digits 37. This format is used directly in some genetic evaluation software for genomic selection.

Listing 37: General syntax of the Exporting

```
snppit --export genotype_set --name=gs_041 -f 0125 -o gs_week32
head gs_week32.0125gen
14941_0 01022002200000
14658_0 00022002200000
66969_9 55555555555551

head gs_week32.0125map
RS01 1 212222
RS02 1 88711
RS03 2 1212456
..
RS14 3 212988
```

The *ped* and *0125* format produce both ASCII text files that can be looked at with any editor provided they are not too big. This can easily happen, as each SNP requires 4 bytes, such as ' *A B*'. With millions of SNPs and thousends of samples the resultant ped file can easily occupy many gigabytes. The *0125* format uses much less space, as each SNP occupies only one character, from the list of *0125*. Accordingly, the file will be around a quarter the size of a ped file for the same data set.

The file extensions for the PLINK formats are *map* and *ped* for the ASCII ped files, and *fam*, *bim*, and *bed* as PLINK requires for the much smaller and faster to process binary format.

The file extensions generated for the *0125* formats are *.0125map* and *.0125gen*, the *gen* standing for genotypes as can be seen in Listing 37. Similarly, for the *ib705* format the file extentions are *.ib705map* and *.ib705gen*.

The bed file still uses less storage as it stored one SNP in two bits only, i.e. one character holds 4 SNPs. As a results, the bed file will be again one quarter the size of the *0125* format and 1/16 of the ped file. Apart from storage space considerations, smaller files require less processing time. Exporting to a bed instead of a ped file typically doubles the processing speed, with the rates for the latter approaching easily 200mio SNPs / sec

for larger panel sizes.

Thus, everything seems to be in favour of using bed file format. This is certainly true if the exported file is to be processed with PLINK or R or WISARD which can directly read bed files. Also their processing speed will be much higher as compared to processing the ped counterpart.

The disadvantages of bed file emerge, when the user wants to look at its content: this cannot be done with editor as its content is non ASCII compressed. Further, if its content is to be further processed down stream by software not already supporting the 2 bit format, adopting this software will be much more complicated than processing ASCII formats.

The ped and bed formats also create a file each that contains map data, i.e. the naming and ordering of the SNPs in the ped and bed file. These are text files that can be viewed with a text editor. The file names have the same base but extension *.ped* and *.map* for the ped format and *.bed* and *.bim* for the binary bed format. The latter further produces a *.fam* file, wich contains the sample IDs and further data. It is actually the first 6 columns form the ped file.

A word of warning: from the above it is clear, that the different export formats as a general do just that: export the same information but in a different format i.e. presentation: *0125* writes a '0' for the first genotype (as provided by the importer, which as such does not have a meaning other that that this is one homozygous genotype). This is different for the *ib705* format as mentioned above: here a '0' is defined as genotype 'BB' which, following the Illumina setup is the defined basepair genotype *ATAT*?????. This has implications: if an export is made with either ped, bed, 0125 for an import that has been made with either of them, the export file will be consistent in that they will only show genotypes, of either AB or *0125* type, assuming nothing about which basepairs they may represent.

If on the other hand *ib705* is used as export format, the 'exporter' may assume that the exported 0125 may also have a meaning on the ATCG basepair level. However, this is only true, if the import has been made for a Interbull 705 file with the *ib705* format. If on the other hand *ib705* is used to export SNPs that were imported with the other formats, then, obviously, the genotypes 012 cannot and and do not have a basepair meaning. These export files are then strictly speaking not Interbull 705 files. So, maybe, *ib705* should be avoided altogether on exports just to avoid confusion if basepair 'meaning' is at all relevant somewhere in the downstream analysis.

If you are only using the exports for gBLUP genomic selection swapping genotype 1 and 2 will not make a difference, and, thus either format *0125* and *ib705* can be used.

4.7.2 Retrieving a sample selection vector

Sometimes it is useful to retrieve the list of sample IDs behind a certain individual selection vector. This can also be done with the -R reporting module. The command and output is given in Table 38

Listing 38: retrieve samples in a sample selection vector

```
1 eg(eno,~): snppit -E individual_selection --name is_002 [-o filename]
2 List Of Selected Individuals from is_002
3 4892
4 4909
5 4944
6 4956
7 ..
```

4.7.3 Retrieving a SNP selection vector

In the same way as sample names can be retrieved, the list of SNP in a particular SNP selection vector can also be produced. This is given in Listing 39.

Listing 39: retrieve SNP names in a SNP selection vector

```
1 eg(eno,~): snppit -E snp_selection --name ss_002 [-o filename]
2  SNPlist from in ss_002
3  rs3094315
4  rs6672353
5  rs4040617
6  ...
7  ...
```

If you know certain SNPs that you would not want to be included in a specific genotype set you can start with such a listing and write them to a file as indicated. Then a text editor can be used to delete those SNPs you do not want.

In the next step you import the edited SNP list into the TheSNPpit using the *-C* action and base your new genotype set on this reduced selection vector.

4.7.4 Retrieving phenotypes

This action exports the phenotypes of individuals of the genotype set specified (Listing 40).

The resulting spreadsheet csv file is identical to that being created during the export of the genotype set, which creates a map, the SNP file and the phenotype csv file.

Listing 40: retrieve phenotypes of a genotype set

```
1 eg(eno,~): snppit -E phenotype --name gs_002 [-o filename]
```

The advantage of using this export function on the phenotypes directly is the faster execution as no possibly large volume SNPs files are not created.

4.8 Report (-R)

The *-R* action facilitates the generation of some reports from the SNP database. In contrast to the *-E* action only meta information with little volume is generated. Therefore, the output is not written to a file but displayed on the terminal instead.

Its general syntax is given in Listing41

Listing 41: General syntax of the Reporting

```
snppit --report|-R  panel|genotype_set|individual_selection|snp_selection\
                   |duplicates|phenotype [suboptions]
        suboptions are:
            --panel   <panel_name>

report -R panel
report -R genotype_set [--panel 770K] --- only GS from panel 770K
report -R individual_selection [--panel 770K]
report -R snp_selection        [--panel 770K]
report -R duplicates            --name gs_002
```

These commands help to keep an overview what is actually loaded int the database.

4.8.1 Report on the panels

The highest level of organization are the SNP panels or name of the SNP arrays. Specific information about all panels in the database can be obtained through the panel command as given in Listing 42.

Listing 42: Report on SNP panels

```
eg@eno:~/database/snp/regression/tmp$ snppit -R panel

                                    List of SNP panels

 Panel   | nSNP   |nSample|SNP(mio)| Timestamp  |           Comment
---------|--------|-------|--------|------------|---------------------------------
P.00200K|200000 | 25000 |  5000  |2015-10-31..|imp.from 00200000-25000.map
P.00100K|100000 | 25000 |  2500  |2015-11-01..|imp.from 00100000-25000.map
P.00054K| 54000 | 25000 |  1350  |2015-11-02..|imp.from 00054000-25000.map
A_B4mio |4098136|  270  |  1106  |2016-06-16..|imp.from gs_009.ss_006.is_007.map
 AB4mio |4098136|  270  |  1106  |2016-06-16..|imp.from gs_009.ss_006.is_007.map
 hapmap |4098136|  270  |  1106  |2016-05-03..|imp.from hapmap2.map
 44444  | 44444 | 14064 |  625   |2015-11-24..|imp.from 00044444-02344.map
   AB   |200000 | 1799  |  359   |2016-06-16..|imp.from gs_005.ss_003.is_005.map
 20SNPs |  20   |  20   |   -    |2016-05-11..|imp.from 00000020-00020.ped
 Total  |   -   | 91693 | 13152  |     -      |
```

The first column gives the name of the panel as it is to be used when referring to it from the TheSNPpit software. The 'nSNP' gives the size of the panel in terms of number of SNPs on the chip. The Timestamp column give the date and time the panel was created or modified. The last column lists the comments inserted by SNPpit and made by the user to supply further information.

If a list of all SNP names in a panel is required you can use the full listing of a SNP selection as described in Listing 39. You have to choose the full snp selection vector which was used during the first import or loading relative to the panel in question.

4.8.2 Report on genotype sets

Genotype sets are the final objects which are used for exports and data analysis. They are completely define by an individual_selection and a snp_selection vector. A corresponding report is shown in Table 43.

Listing 43: Report on genotype sets

```
1  eg(eno,~): snppit -R genotype_set
2  eg@eno:~/database/snp/regression/tmp$ snppit -R genotype_set
3
4                                    List of Genotype Sets
5
6  SetName|IndSel|SNPSel| Panel  |   Timestamp   |               Comment
7  -------|------|------|--------|---------------|-----------------------------------
8  gs_001 |is_001|ss_001|P.00054K|2015-10-31 .. |[IDs only from 054000-25000.ped]
9  gs_002 |is_002|ss_001|P.00054K|2015-10-31 .. |[All IDs  incl 054000-25000.ped]
10 gs_003 |is_003|ss_002|P.00100K|2015-10-31 .. |[IDs only from 100000-25000.ped]
11 gs_004 |is_004|ss_002|P.00100K|2015-10-31 .. |[IDs only from 200000-25000.ped]
12 gs_006 |is_006|ss_003|P.00200K|2015-10-31 .. |[All IDs  incl 200000-25000.ped]
13 gs_007 |is_002|ss_004|P.00054K|2016-02-19 .. |[Chroms:except W with 54000SNP]
14 ..
15 gs_009 |is_007|ss_006| hapmap |2016-05-03 .. |[IDs only from hapmap2.ped]
16 gs_010 |is_008|ss_007| 20SNPs |2016-05-11 .. |[IDs only from 00000020-00020.ped]
17 gs_011 |is_009|ss_008| AB4mio |2016-06-16 .. |[IDs from gs_009.ss_006.is_007.ped]
```

Sometimes you will want to work only with genotype sets from a certain panel, then the panel name needs to be specified as shown in Listing 44.

Listing 44: Report on genotype sets within a panel

```
1
2  eg@eno:~/database/snp/regression/tmp$ snppit -R genotype_set --panel P.00054K
3
4                            List of Genotype Sets for panel P.00054K
5
6  SetName|IndSel|SNPSel| Panel  |   Timestamp   |               Comment
7  -------|------|------|--------|---------------|-----------------------------------
8  gs_001 |is_001|ss_001|P.00054K|2015-10-31 ... |[IDs only from 054000-25000.ped]
9  gs_002 |is_002|ss_001|P.00054K|2015-10-31 ... |[All IDs  incl 054000-25000.ped]
10 gs_007 |is_002|ss_004|P.00054K|2016-02-19 ... |Chroms: only W with 54000 SNP]
```

4.8.3 Report on sample or individual sets

For each genotype set an individual_selection vector is specified. All those sets or selections can be retrieved as shown in Table 45.

Listing 45: Report on sets of individuals

```
eg(eno,~): snppit -R individual_selection
           List of individual selections
 Panel   |IndSel |nInd|Source |        Comment
---------|-------|----|-------|----------------------
CHICK68K|is_003 |  48|Initial|Complete Genotype Array
 GWA-NP |is_006 |2000|Initial|Complete Genotype Array
 PPARG  |is_007 |4000|Initial|Complete Genotype Array
 amylo  |is_001 | 404|Initial|Complete Genotype Array
 hmap2  |is_004 | 270|Initial|Complete Genotype Array
koell-1 |is_005 | 144|Initial|Complete Genotype Array
  wgas  |is_002 |  90|Initial|Complete Genotype Array
ck-na32 |is_008 |5317|Initial|Complete Genotype Array
        |-is_009|  60|is_008 |Hungarian Samples
```

Often, sets will be derived from already loaded sets in the database. The *is_009* is one of those. From the initial set *is_008* of 5317 samples a new subset from one country has been defined comprising 60 individuals. That it has been derived from a set already in the database is indicated by the column *Source* which notes *is_008* and graphically by the indention in column IndSel. In this way it can be easily seen which sets are subset of which parental.

If the content of the individual_selection vector is to be retrieved, i.e. the actual ID of the individuals or sample the *--export* has to be used (see 4.7).

4.8.4 Report on SNP sets

A SNP set is the second vector that is required for the definition of a genotype set. A listing of them is given in table 46.

Listing 46: Report on sets of snp selections

```
eg(eno,~): snppit -R snp_selection
                List of SNP selections
 Panel   |SNPsel | nSNP   |Source |        Comment
---------|-------|--------|-------|----------------------
CHICK68K|ss_003 |   57636|Initial|Complete Genotype Array
 GWA-NP |ss_006 | 317503|Initial|Complete Genotype Array
 PPARG  |ss_007 |     41|Initial|Complete Genotype Array
 amylo  |ss_001 |   36455|Initial|Complete Genotype Array
 hmap2  |ss_004 |4098136|Initial|Complete Genotype Array
koell-1 |ss_005 |   64232|Initial|Complete Genotype Array
  wgas  |ss_002 | 228694|Initial|Complete Genotype Array
        |-ss_008| 123999| ss_002| only 10 chromosmes
```

The information provided in this report is analogous to that in Listing 45

Depending on the option chosen, lists are generated as shown in Listing 45. Again, the indention reflects the same subsetting as for the sample report.

4.8.5 Report on phenotypes

With the EAV (Entity-Attribute-Value) model, that has been implemented in TheSNPpit, any number of phenotypes or traits or other attributes can be stored for samples already stored in the database. Through a header line in a comma separated file new 'phenotypes' can get easily created. If not done prudently, different spellings for the same attribute may get created in the database.

Listing 47: report phenotype names for a given genotype_set

```
eg(eno,~): snppit -R phenotype --name=gs_001

   List of phenotypes from gs_001
      Phenotypes|count
      ----------|-----
      bf        |   54
      Bf        |    1
      born      |    4
      breed     |    8
      dam       |    3
      dg        |    5
      fce       |    5
      nba       |    5
      sire      |    3
eg(eno,~)
```

As can be seen, in this individual_selection which defines the samples in the genotype set *gs_ 001*, in all nine different phenotypes have been defined. Apparently, there is one mistake in data entry: we have, both, *bf* and *Bf*, that latter probably a typo.

If the actual phenotype data are to be retrieved, use the export of *gs_ 001*. Along with the map and genotype data also the phenotype data are export in a csv file.

4.8.6 Report duplicates

As has been described in 5.2.1, it is possible to import more than one genotype record per sample. It is the obligation of the user to decide if this should be done.

Listing 48: retrieve duplicates

```
1  eg(eno,~): snppit -R duplicates --name gs_001
2
3
4             Sample_Name|count|sample_id
5             -----------|-----|---------
6              V0000005  |3    |2
7              V0000004  |2    |1
8              V0000006  |2    |3
9
10                      List of individual duplicates
11
12 Sample_Name|    time_stamp     |geno_data_id|         differ?
13 -----------|-------------------|------------|-------------------------
14  V0000004  |2015-11-24 19:23:05|     1      |genotypes identical
15  V0000004  |2016-01-23 09:12:09|     26     |genotypes differ:  40 / 2
16  V0000005  |2015-11-24 19:23:05|     2      |genotypes identical
17  V0000005  |2016-02-14 12:05:05|     22     |genotypes identical
18  V0000006  |2015-11-24 19:23:05|     3      |genotypes identical
19  V0000006  |2015-11-24 19:23:05|     23     |genotypes differ:  40 / 5
```

As shown in listing 48 this command checks for duplicates for a given genotype set. It can be used to check the genotype sets and possibly clear up things. There are two parts of the output.

The first lists all those sample with multiple of more than 2 records in the table *genotype_ data.* It can probably be assumed that a database with no entries here is in good shape: for a given panel (genotype sets are defined for a given panel) only one set of SNPs should have been generated unless a sample has been re-genotyped. Thus, with no lines in this part of the Listing, all samples have only one set of genotypes: thus, all is well! In this situation, the fastest database interaction using binary cursor will automatically be chosen, and then exports are as fast as it gets, possibly more than 250 million SNP/second!

The second table in Listing 48 gives details on each individual import for a given sample name with more than 1 genotype record. The genotypes vector can either be identical or different as shown in the last column. Again, the user needs to investigate and decide what to do about it. If one record is wrong, then deleting it may be appropriate. Here, the *genotype_ data_ i*d is added which is required for deleting the genotype record with TheSNPpit. Sorting of this part of the output is by sample name and time stamp. The latter is intended to track down the insertion date of the genotype record. In this example, it turns out that the genotypes for *V000005* got imported the first time in 2015, while the others were inserted in 2016. This might ring a bell with the user: '*2015-11-24 was a test run*'. As the two records provide identical SNP information either of them and can be deleted using the unique ID number in column *geno_ data_ id.*

Line 14/15 in Listing 48 shows an example with some of the SNPs are different: for *V0000004* two of the 40 SNPs are different, while 5 are different for sample ID *V0000006* in line 19. Again, the user has to make up her mind which is correct and which should be deleted. The reason for such an duplication with different SNPs can be retyping.

These duplicates can be avoided, if an import is run first with the *--dryrun* option.

But also here: while information is provided, only the user can resolve the clash if there are duplicates in the new dataset when imported to the database.

There are two steps in checking for duplicates: with *--dryrun* first the input file will be scanned for duplicate sample names and occurrences be reported. In the second step the sample names are checked against an occurrence in the database for the given panel. As the *--dryrun* implies: no data will loaded into the database, only checking is performed.

4.9 Batch mode

Often it is desirable to have a number of steps executed from a file, if only to remember what the actual command were. In principle, this is simple: just put the snppit program command line invocation in text file like *batch1.txt* and then execute it in the bash by typing '*bash batch1.txt*'. Then all commands will be executed one after the other until the last line in the *batch1.txt* file has been reached. In this way the commands in Table 49can be executed.

Listing 49: execute snppit commands from a file

```
eg(eno,~): cat batch1.txt
#!/bin/bash
# list all panels
snppit -R panel
snppit -R snp_selection
# output animal IDs:
snppit -E individual_selection --name is_002 -o samples_from_is002.txt
eg(eno,~): bash batch1.txt
                          List of SNP panels
    Panel    |Platform| nSNP |                  Comment
-------------|--------|------|-----------------------------------------
  breeds-na33| PLINK  |580961|imported from PLINK-na33.map
  duck600-32 | PLINK  |580961|imported from na33.map
    amido    |  ospr  | 36455|imported from ECocData_78.txt

                           List of SNP selections
    Panel    |SNPsel| nSNP |Source |               Comment
-------------|------|------|-------|-------------------------------------
    amido    |ss_001| 36455|Initial|Complete Genotype Array
 duck600-32  |ss_007|580961|Initial|Complete Genotype Array
             |ss_008|580961|Initial|Complete Genotype Array
  breeds-na33|ss_002|580961|Initial|Complete Genotype Array
eg(eno,~)
```

It is good practice to include comments with your commands. In that ways you can later on remember what you have done.

This simple setup, unfortunately, does not always work. Often, the following command requires the output of the previous. For instance, when a new genotype set is to be created on the basis of a new snp_selection two steps are required. First, a new SNP selection is created, for instance by importing a list of SNPs. This will lead to a new SNP selection vector being created, whose name is returned by the program call, e.g. ss_012 .

In the second step the new genotype_set is created on the bases of an existing individual selection already in the database (is_009) and the newly created ss_012 . This

means that no static sequence of commands in a batch file will do the trick. Instead there has to be a mechanism to pass the newly created SNP selection name on to the second call.

Table 50 shows the sequence of steps when performed manually on the command line.

Listing 50: Manually creating a new genotype set

```
1 eg(eno,~):snppit -C snp_selection -x ss_019  -p amido -i Wsnp.txt -c 'W chrom.'
2    snp selection ss_021 created with 23021 SNPs
3 eg(eno,~):snppit -C genotype_set --n ss_021 -s is_009 -c 'W Chrom. & 388 samples'
4    genotype set gs_005 created from ss_021 and is_009
5 eg(eno,~)
```

Clearly, the user first needs to execute line 1 to obtain the *ss_021* and then use this as input in the second. This, of course, means that the script cannot be executed in one series of steps.

This problem can be solved by a bash script as shown in table 51. This Listing contains the same commands as the previous one.

Listing 51: bash script for creating a new genotype set

```
1 #!/bin/bash
2 SS=$(snppit -q -C snp_selection -x ss_019  -p amido -i Wsnp.txt -c 'W chrom')
3 GS=$(snppit -q -C genotype_set -n $SS -s is_009 -c 'W Chrom. & 388 samples')
```

The only difference is the prepended "*SS=$(*". The *SS* is a bas variable and holds the snp selection setname after the command in brackets has been executed. This is then passed to the second call to snppit with the variable "*$SS*". The *-q* is a switch for *quiet* which means that no intermediate output is printed except for the set name which is collected in the bash variables *SS* and *GS*.

With this mechanism longer scripts can be built to execute pipelines automatically, with dynamically creating and using individual and snp selection vectors as well as genotype sets.

5 Considerations

In this paragraph we shall present issues that are important and have not been sufficiently delt with before. The user is advised to also read these lines, because they may point to problems that can arise should then not be addressed. Some of the issues have been mentioned before, but duplication is better than omission.

5.1 Consistency of SNP data files

As noted above, the very common PLINK file formats are the default import and export formats in TheSNPpit. The SNP genotype data are stores in the *.ped* file with one line

per sample starting with the sample ID and from column 7 on one SNP genotype after the other. For a 600K panel, each line will be very long with these 600000 genotypes. As this is the only information in the ped file we do not know which actual SNP is in any one position. This is noted in the map file which has basically one line for each SNP. Thus line 1214 in the map file gives the SNP name and details on the genotype stored in the position 1214 in the ped file.

Now imagine, that the order of line in the map file is somehow changed from what it should be: then line 1214 may have a changed for instance by swapping lines 1214 and 1215. As a result, the SNP in position 1214 gets the name from line 1215 and vice versa and the genotype is actually for a different SNP than you think. Such a swap will only affect 2 SNPs, but imagine what happens if the last line from the map file gets deleted at the end and inserted as the first: all SNPs will be wrong. On import you will get a mixture of correctly and wrongly annotated SNPs.

This to happen is a real nightmare as you will probably never really find out, only your results may be funny or not in agreement with others.

So clearly, utmost care needs to be taken by the user to ensure that the mapfile actually describes the ped content. The user has to positively determine, that either each batch received from the genotyping lab has the SNPs in the same order, or that each ped file is always accompanied by the appropriate map file. There is no way of identifying a mismatch during import! Only your Nature paper will be based on wrong position data of your SNPs. BTW: this issue applies to all input formats.

I hope that this has sufficiently scared you!

This said: once you positively know that the order of the SNPs does not change from one batch of files to the next you can of course use only one map file for all imports. But as TheSNPpit requires a map file with the same basename you can conveniently make a link to the original map file: '*ln -s 600Korig.map week22.map*' if you want to load the ped file *week22.ped*. As an added measure of security in avoiding the described poblem this has to be done manually for each import when a new filename is used.

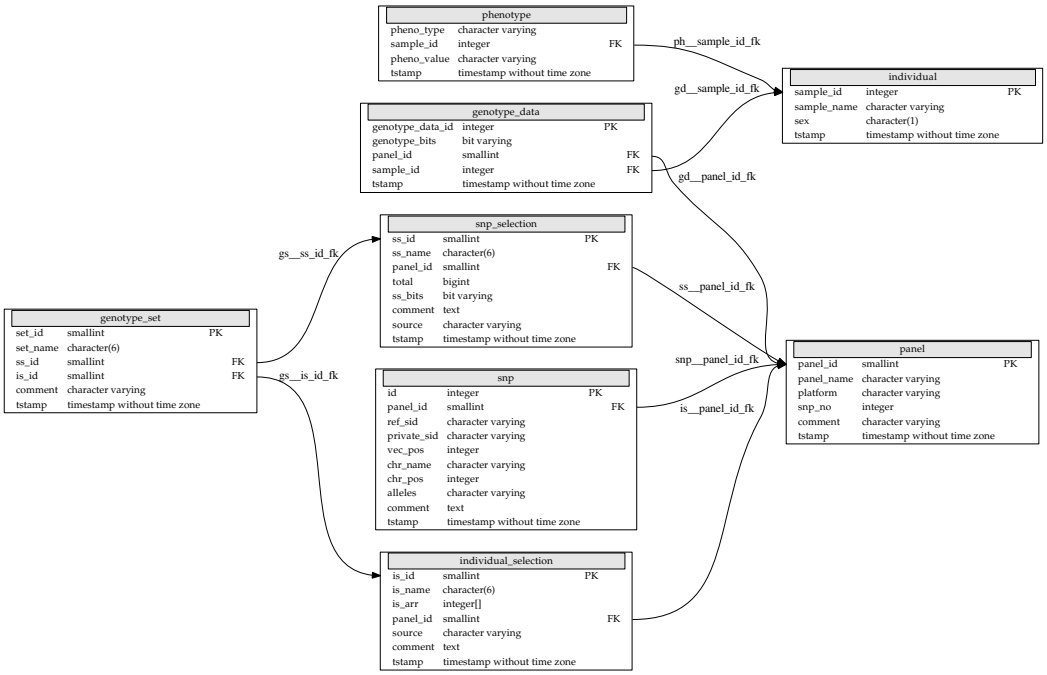
There is one other issue, that might lead to problems. In the PLINK format, the *A* genotype is defined as the major allele, which means that the value in one sample is a function of the others in that batch. Also, verify positively, that the definition of the *A* genotype does not change from one batch to the next.

5.2 Sample name issues

The sample name is of central importance in TheSNPpit as it connects the outside world, e.g. a given animal with genotype records with the content in the database. It was the strategic decision of the developers of TheSNPpit to allow full flexibility in terms composition and use of the sample name. This means that from the software side no constraints are imposed on the sample IDs. In particular the same sample name can be used repeatedly and it is the obligation of the user to avoid using the same ID for different samples.

As a data management backend TheSNPpit is intended to be used for multiple panels of any size and as such can of course be used for samples from any species. It therefore

makes a lot of sense, that a user specifies her policy regarding sample ID definition prior to populating the database. In the following we shall outline implications of certain choices.



5.2.1 Import

The sample name given in the ped file is considered a unique name in the complete database across all panels. As can be seen from the ER diagram for tables *individual* and *genotype_data*, the *sample_ID*, is a sequence and a one to one relationship with the *sample_name*, i.e. the name supplied by the user. This setup has the following implications:

1. if a new sample name is read from the import genotype file, a new record is created in the *individual* table with a new sequence ID.

2. if a genotype records is to be imported for a *sample_name* already existing, its *sample_ID* is used in the table *genotype_data*. Thus, multiple genotype records can be created for one *sample_name*! As a result you can have multiple SNP records from one animal even from different panels.

The reasons for not restricting the storage of genotypes are:

1. samples may get genotyped for more than one SNP panel, thus one sample will have more than one genotype.

2. an animal or individual may have gotten retyped with the same panel, and it should be possible to store this data as well.

3. there maybe imputed SNPs which originated in a smaller panel but are to be used together with originally genotypes data.

This has some implications for the definition of sample names. If we are dealing with the international 12 digit cattle ID, then there will be no problem with the same name referring to different animals: it will simply not happen, as they are unique world wide.

If on the other hand three digit/character names are use from an experiment and results from more than one experiment are to be stored in the same database, then it may make sense to prepend some experiment identifier to the actually used name/ID: *exp1-123*.

There is always the option of setting up separate databases for different species/uses.

5.2.2 Implications for genotype set

Genotype sets, i.e. the unit for export, are defined on the basis of a SNP selection vector and a list of sample names within a given panel. Here having multiple genotype records for one sample name needs consideration. At this stage we shall leave out the issue of using the same name for different individuals, which most certainly has to be avoided by the user. However, having multiple records within a panel for one individual can certainly make sense, as seen above. But what are the implications for a genotype set? A genotype set is exported using this simple algorithm:

1. read the snp selection vector pertaining to the genotype set to be exported
2. read the individual selection vector pertaining to the genotype set to be exported
3. read sample IDs from the individual selection vector
    a) fetch the genotype vector for the sample ID
    b) shrink the SNP vector according to the snp selection vector
    c) write to file
    d) read next sample ID
    e) goto a) until done

This is the normal setup if one sample ID has only one genotype record. We would then export as many genotype records as there are entries in the individual selection vector: 52 entries in the selection vector and 52 exported genotype records.

If however, more than one genotype record has been imported at various stages, the action 3.a) from the above pseudo code will not only return one but multiple genotype records just depending on how many had been imported before.

5.2.3 Duplicates and processing speed

The genotype records are transfered from the PostgreSQL database through what is called a cursor. Given the database design, independent of the type of cursor used

TheSNPpit will beat all other systems that we know of hands down. Our first proof of concept implementation in Perl was already a factor 10 faster than the published data from other implementation. After a rewrite of the database access in C we are now about 100 times faster than our already very fast Perl implementation. However, even at this level quite a bit can be gained by choosing the binary cursors versus the standard cursor version. If however, multiple genotype records exist for one sample ID and we want to process all of them then the fastest binary cursor cannot be used but using the standard instead.

Thus, prior to an export or other access on the basis of genotype sets, the maximum number of duplicates is determined and the appropriate cursor chosen. The listing 52 show the output from two runs.

Lines 1 through 14 are the output from exporting the initial import which resulted in the genotype set *gs_001*. Then three more records were added to *gs_001* as shown in listing 52 on line 16. This created a new genotype set adding the three new genotype records the *gs_001* resulting in *gs_004*. Here we need to add, that the three 'new' records had actually been imported already when creating *gs_001*. Thus, line 16 did add three more genotype records but did not add three new sample ID because they had already been stored.

The effect on the export can bee seen in the last part of listing 52. Firstly, the duplicate ID are listed as shown in lines 18–23.

Listing 52: effect of duplicate genotype records

```
1  ==================================================
2  Export finished, written to file gs_001.ss_001.is_001.ped
3  using module        binary cursor
4  ----number of sample_names processed : 90
5  ----highest # of SNP recs per sample : 1
6  ----number of genotypes written      : 90
7  ----number of SNP per sample         : 228694
8  ----number of SNP flipped            : 0
9  ----total number of genotypes written:    20.582 million
10 ==================================================
11 ==================================================
12 No. of genotypes processed        :    20.582 million
13 Number of SNP processed per second: 112.70 million
14 ==================================================
15
16 snppit -A gs_001 -f ped -i add3.ped
17
18 1 - dupli. geno for sample_name NA18526 and sample_id 1
19 2 - dupli. geno for sample_name NA18524 and sample_id 2
20 3 - dupli. geno for sample_name NA18529 and sample_id 3
21 ==================================================
22 Export finished, written to file gs_004.ss_003.is_004.ped
23 using module               cursor
24 ----number of sample_names processed : 90
25 ----highest # of SNP recs per sample : 2
26 ----number of genotypes written      : 93
27 ----number of SNP per sample         : 228694
28 ----number of SNP flipped            : 0
29 ----total number of genotypes written:    21.269 million
30 ==================================================
31 ==================================================
32 No. of genotypes processed        :    22.269 million
33 Number of SNP processed per second:  52.19 million
34 ==================================================
```

You should check if this really what you intend to do. Secondly, we see that we have for at least one sampleID a max of 2 SNP records, actually here for each of the three. As it is our policy to export all SNP records belonging to a sample ID (within a panel) we have now 93 genotypes written to the file as seen in line 26 of listing 52.

Finally, lets look at the export performance. Line 33 give us an export of 53 mio SNP per second. This already staggering number is more than doubled if we do not have to handle duplicates: line 13 gives the figure 112 million per second (that's more than most humans can do manually).

5.2.4 Removing duplicates

As we have seen above, removing duplicates speeds up the data exchange with the database considerably. Thus, if they can be avoided, then processing speed will be much improved. If you do not really need any of the duplicates in your database for exporting and filtering, you can simply produce an individual selection vector that does not include the duplicates.

5.3 Deletions

The delete action is a very innocent looking little statement which can basically through its cascading delete action wipe your database. This should convince you, in addition to the standard disc crash concern, that you should have proper up to date backup. Only then will you be able to recover from a loss of any sort.

Inappropriate use of some delete actions may find you all of sudden without any genotype sets in the database. Then you would not be able to export anything. In this situation not all is lost provided you have not deleted actual genotype data. Assume that you delete the snp selection vector for the initial load of a genotype data file. Then, obviously, you will also not have a genotype set which was based on the complete snp selection vector.

You can create a new complete SNP selection vector as describe above in chapter "*Create snp selection*".

6 Installation

TheSNPpit is a Perl package with compute intensive tasks written in C with PostgreSQL being used as the database backend. It requires a number of additional packages which are freely available but have to be configured correctly to work. Thus, the complete system can be configured free of licensing costs.

6.1 Hardware and operating system requirements

SNP data are potentially huge in volume. Along with large data volume also processing time may be consideration. A typical system would be any Intel or AMD 64bit system with may be 8 or more GB of RAM and a sizable hard disk. Here a SSD will really speed up operations. Because of the exceptional speed of TheSNPpit the export will saturate a normal hard disk meaning that TheSNPpit export will be limited by the speed data can be written to the hard disk. With an SSD this limit will be extended and, accordingly, the export speed be faster. Exporting data from high density panels can easily be GBs, 15GB being no rare case.

A Linux installation, preferably a Debian based, is well suited, easing the installation of TheSNPpit: the following instructions are Ubuntu i.e. Debian based.

6.2 Automated installation for Debian/Ubuntu

On the TheSNPpit download website a bash script is provided which should perform an automated installation of the all required system software components and the SNPpit software. It will further configure the PostgreSQL database, create the required users and groups.

The script must be run with root access from the user root. Clearly, the computer must have Internet access for possible software downloads.

6.2.1 Installation

An automated install script in Listing 53 is available which should run on any Ubuntu/Debian system. In the first steps all required system software components will be installed from the Internet. It does require a PostgreSQL version 9.3 and up being available for the Linux distribution where the install script is executed.

The installation script is written such, that also an already
existing postgresql can likely by used.
However, if you run into issues in connection with 'ports'
completely remove the current postgresql installation and
start again with INSTALL, it will install the postgresql
by itself.

Listing 53: Software installation

```
1
2 # as user root:
3 root>cd /usr/local
4 root>wget --no-check-certificate \
5      https://tsp-repo.thesnppit.net/download/TheSNPpit-latest.tar.gz
6 root>tar xzvf TheSNPpit-latest.tar.gz
7 root>cd TheSNPpit-0.1.23 (or whatever the downloaded version is)
8 root>bin/INSTALL
```

The install script performs the following actions:

1. install the system software components: libraries, PostgreSQL
2. creates the Linux user *snpadmin* and the group *snp*
3. configures PostgreSQL
   a) creates PostgreSQL user *snpadmin* with superuser rights
   b) gives user *snpadmin* access rights in *pg_hba.conf*

The script can be run repeatedly in without doing any harm to a running installation. Look at the terminal output to check for errors.

The standard PostgreSQL installation places the database into */var/lib/postgresql* which is in default mode in the root partition. So if you set up a new system you may want to make the root partition large enough. Alternatively, the database can be put in a file system location as described in 6.4.

The default TheSNPpit software installation is */usr/local/TheSNPpit_current*. This is a link to the latest TheSNPpit installation which comes as a tar ball and after extraction creates in */usr/local* the directory *TheSNPpit-1.xx* .

Listing 54 is a copy of the *README.install* file that is supplied with te installation.the directory

Listing 54: Software installation

```
1
2
3  HOWTO install TheSNPpit:
4
5  The installation procedure has been developed for Debian/Ubuntu
6  distributions.
7
8  Prerequisites:
9  The installation ...
10     * has to be done as root
11     * has to be done in /usr/local
12 Note:
13 The installation is written such, that also an already
14 existing postgresql installation can likely by used.
15 However, if you run into issues in connection with
16 'ports'  completely remove the current postgresql installation and
17 start again with INSTALL, it will install the postgresql by itself.
18
19 Here we go:
20
21 1. Become root
22 2. cd /usr/local/
23 3. wget --no-check-certificate \
24    https://tsp-repo.thesnppit.net/download/TheSNPpit-latest.tar.gz
25 4. tar xzvf TheSNPpit-latest.tar.gz
26 5. cd TheSNPpit-1.23 (or whatever the latest version is)
27 6. bin/INSTALL (run from a terminal window)
28
29 The following steps are performed by bin/INSTALL:
30
31 - install all required system components like Perl, PostgreSQL, gcc
32   and a whole host of libraries
33 - the user snpadmin and group snp is created and the software beneath
34   /usr/local/TheSNPpit_current is owned by snpadmin and can be executed by
35   members of the Unix group snp.
36 - PostgreSQL is configured
37 - the database is created and a test run performed
38 - after the installation, a demo can be run by executing bin/demo.bat (usually
39   by root, depending on your Postgresql access rights, needs right to create
40   database)
41 - bin/INSTALL and bin/demo.bat can be run several times. They are
42   non-destructive
43
44 To create TheSNPpit users other than root consider these commands on the
45 prompt of your terminal (replace <your_user> with your username):
46
47 * adduser <your_user>       # create user on OS level (if not existing)
48 * adduser <your_user> snp   # add your user to Unix group snp
49 * createuser <your_user>    # create database user (read the fine manual
50                              # 'man createuser' for the appropriate access
51                              # rights)
52 * log into <your_user>
53 * put in the ~/.bashrc the search path of <your_user>:
54   export PATH=$PATH:/usr/local/TheSNPpit_current/bin
55 * log out and in again
56 * run the demo:
57   cd /tmp
58   demo.bat
59
60 Enjoy!
```

6.2.2 Creating the production SNPpit user

The user performing the database accesses is the one that also owns the input and output data. As such it will be the Linux user. To provide database access to the (existing) Linux user *goofer* it will have to be created also as a PostgreSQL user. Additionally, *goofer* has to be member of the unix group *snp* to get access rights to the software TheSNPpit.

1. *sudo adduser goofer snp* – adds the user *goofer* to the group *snp*
2. *sudo createuser -s goofer* – creates the database user *goofer*

Beware, the '*-s*' provides administration rights to this user meaning that the user can delete databases (through *dropdb*) including the production database. Thus, for security reasons it might not be desirable to give a user this right. However, there are also good reasons for providing a user with the capacity to create and drop databases. Running the demo is one such possibly very desirable task. The database administrator need to take a decision here and balance security and flexibility.

`http://www.pateldenish.com/2014/02/preventing-human-errors-in-postgres.html` provides a way to secure a (production) database from being dropped. It turns a database into a template database which can not inadvertantly be deleted unless it is turned into a non template database. How this can be done is described in the above document. The database admin should however notice that this template solution does have disadvantages. Read the discussion in the above URL.

6.2.3 Running the Demo

Once the installation is done, the user might want to immediatly create a SNPpit database and execute some commands against it. To help the user get the program call and the parameters correct, the user may execute the script *demo.bat* which is part of the installation. (At this stage we are assuming that you are still logged in as *root*). This script creates the database *TheSNPpit_ demo*, imports data and executes a number of commands which are printed on the screen for the user to read, possibly modify and send to the database.

The demo is started by typing a a console *demo.bat*. Its actions are given in Listing 55.

Listing 55: Running the demo

```
1
2
3  eg@root:~/database/snp$ demo.bat
4  INFO:  Creating TheSNPpit Demo Database ...
5  INFO:  ************************************************************
6  INFO:  *      load panel map data from file picken.map             *
7  INFO:  *          and give it the name chk-57Illu                  *
8  INFO:  *                                                           *
9  INFO:  * Parameters:                                               *
10 INFO:  *    -T TheSNPpit_demo => use demo database                  *
11 INFO:  *    -I panel          => insert new panel                   *
12 INFO:  *    -f ped            => use PLINK format                   *
13 INFO:  *    -p chk-57Illu     => panel name                         *
14 INFO:  *    -i picken.map     => input file                         *
15 INFO:  *                                                           *
16 INFO:  * Hit RETURN to run command:                                *
17 INFO:  ************************************************************
18
19 snppit  -T TheSNPpit_demo -I panel -f ped -p chk-57Illu -i picken.map
20 (waiting for user hit return)
21 ...
22 ...
23
24 INFO:  ************************************************************
25 INFO:  *         load SNP data for the panel chk-57Illu           *
26 INFO:  *                   from file picken.ped                    *
27 INFO:  *                                                           *
28 INFO:  * Parameters:                                               *
29 INFO:  *    -T TheSNPpit_demo => use demo database                  *
30 INFO:  *    -I data           => insert data for panel              *
31 INFO:  *    -f PLINK          => use PLINK format                   *
32 INFO:  *    -p chk-57Illu     => panel name                         *
33 INFO:  *    -i picken.ped     => input file                         *
34 INFO:  ************************************************************
35
36 snppit -T TheSNPpit_demo -I data  -f ped -p chk-57Illu -i picken.ped
37 (waiting for user hit return)
38
39 INFO:  ************************************************************
40 INFO:  *                      list all                             *
41 INFO:  *  genotype sets, snp_selection and individual_selection    *
42 INFO:  *                   in the database                         *
43 INFO:  *                                                           *
44 INFO:  * Parameters:                                               *
45 INFO:  *    -T TheSNPpit_demo => use demo database                  *
46 INFO:  *    -R genotype_set   => print report of genotype_sets      *
47 INFO:  ************************************************************
48
49 snppit -T TheSNPpit_demo -R genotype_set
50 ...
51 ...
```

Once the demo script is finished, the database is still available for the user to play around with. This script can also be run under the user account provided the user has been activated as a database user as shown from line 48 on in Listing 54.

The demo database can be used for further playing around. Start snppit with the "--

man" flag to obtain examples of commands which can be tried out on the demo database.

6.2.4 Creating the production database

A production database or any other can be created through the following steps:

1. *createdb TheSNPpit*
2. *psql TheSNPpit -f /usr/local/TheSNPpit_ current/lib/TheSNPpit.sql*

Hereafter an empty database with the name *TheSNPpit* is readily available for use. The command "*psql -l*" will list all PostgreSQL database on the system as shown in Listing 56.

Listing 56: Create database TheSNPpit

```
1  eno~>createdb TheSNPpit
2  eno~>psql TheSNPpit -f /usr/local/TheSNPpit_current/lib/TheSNPpit.sql
3   some output
4   ...
5
6  eno~>psql -l
7                                    List of databases
8     Name     |   Owner   | Encoding |   Collate   |    Ctype    |
   Access privileges
9  -----------+-----------+----------+-------------+-------------+-----------------------
10  TheSNPpit | eg        | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
11  postgres  | postgres  | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
12  template0 | postgres  | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres
   +
13            |           |          |             |
   | postgres=CTc/postgres
14  template1 | postgres  | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres
   +
15            |           |          |             |
   | postgres=CTc/postgres
16 (4 rows)
```

It may be very convenient to have a test database. TheSNPpit accesses the database with the name *TheSNPpit_ test* by adding the *-T* flag to the program invocation. It can be created just as shown in Listing 56.

6.3 Manual Installation on other Linux Distributions

Users who have installed TheSNPpit through the install script can skip this paragraph, while this may be useful for installations on non Debian based Linux distributions.

In the following the installation steps will be described for a Debian based Linux system. The steps that are given in detail here are actually the ones performed in the automated installation script. There is no reason to assume, that TheSNPpit cannot also be run on other Linux distribution, only the software installation commands may vary: instead of the *apt_ get* you will use *rpm* on a Redhat installation and *yum* on a SUSE distribution. Also, the package names might vary.

On a Redhat system you need to install the package shadow-utils to get the adduser/addgroups commands.

Clearly, to perform the complete installation you need to have root access. For a complete installation of TheSNPpit on your Linux machine you need to install and configure a number of freely available packages. These are the configuration issues:

1. system software installation
2. database configuration
3. TheSNPpit software installation
4. testing

6.3.1 System software installation

On a Debian or Ubuntu machine the following commands may work:

Listing 57: Software installation

```
1 apt-get install perl gcc
2 apt-get install PostgreSQL PostgreSQL-contrib
3 apt-get install libecpg6 libecpg-dev
4 apt-get install libdbi-perl libinline-perl libmodern-perl-perl
5 apt-get install libcloog-ppl1 libcloog-ppl0
6 apt-get install libfile-slurp-perl
7 apt-get install libdbd-pg-perl
8 apt-get install libjudy-dev
```

If you have a Redhat or SUSE system you will need to use the appropriate package installation program like *rpm* and *yum*. Remember, installation of system software has to be done as the root user, not with the login of a user who later uses the system. If you do not know what I am talking about: then you should get help from your system administrator.

6.3.2 Database configuration

This setup assumes, that PostgreSQL runs on the same computer as TheSNPpit software. Furthermore, we assume wide open access from the user to the database. If you do not like this: there are all sorts of access controls available. But this involves detailed knowledge, so if you want it: dig in!

Listing 58: Database configuration

```
1  # 1. Configure file /etc/PostgreSQL/9.4/main/pg_hba.conf
2  #    the file should contain these lines:
3  host  all   snpadmin   127.0.0.1/32   trust
4  host  all   snpadmin   ::1/128        trust
5  # 2. restart PostgreSQL (as root):
6  service PostgreSQL restart
7  # 3. need to do the following as user 'postgres'
8  su - postgres
9  # 4. create database users
10 createuser --superuser eg
11 createuser --superuser snpadmin
```

1. You need to be super user to access the file *pg_ hba.conf* . You will find the preinstalled configuration at the end of that file. Comment out the installation default by a '#' and insert those two line instead.

2. After saving the file, PostgreSQL needs to get restarted as indicated in line 7 of listing 58.

3. Next, the database users need to be created for PostgreSQL, which is different from the Linux user. To be able to do this, you need to switch to the user 'postgres' as shown in line 9.

4. Firstly, the user with the Linux account who wants to access the database needs to be specified. As shown in line 11 this would be *eg* for me (Eildert Groeneveld). You need to specify your own Linux login name. Then also the user *snpadmin* needs to be defined. The latter is the database user in the TheSNPpit software.

6.3.3 TheSNPpit software installation

In this step TheSNPpit software needs to get installed. Actually, we can point to the Listing 57 and the text that goes with it in the Installation paragraph. You should be able to execute lines 3 through 6. But the *INSTALL* in line 7 may not run through. But just give it a shot. You can try running the demo as described in the "*Running the Demo*" paragraph

6.3.4 Final steps and testing the installation

A few more steps are required to finish the installation and ensure everything works as expected. The proof of the pudding is in the eating. After the installation you need to check if the database is operational.

The database name is assumed to be TheSNPpit for the production data and TheSNPpit_test for testing purposes. The latter should never contain any data of value, as some scripts will delete and recreate this database for testing. There are a few options for verifying the installation.

Listing 59: Final steps and checks

```
1
2  # 1. test in TheSNPpit programs are accessible from command line
3
4  eg@eno:~/database/snp$ snppit -v
5
6      TTTTTT HH    HH EEEEEE   SSSSS  NN    NN PPPPPP    PPPPPP  II TTTTTT
7        TT   HH    HH EE      SS      NNN   NN PP   PP   PP   PP II   TT
8        TT   HHHHHHH EEEE     SSSSS  NN N NN PPPPPP    PPPPPP  II   TT
9        TT   HH    HH EE          SS NN  NNN PP        PP      II   TT
10       TT   HH    HH EEEEEE   SSSSS  NN   NN PP        PP      II   TT
11
12     -------------------------- VERSION 1.09 ----------------------------
13
14 # 2. database access:
15 eg@eno:~/database/snp$ psql -l
16                                 List of databases
17       Name      |    Owner    | Encoding |   Collate   |    Ctype    |
   Access privileges
18 ----------------+-------------+----------+-------------+-------------+---
19  TheSNPpit      | eg          | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
20  TheSNPpit_demo | snpadmin    | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
21  TheSNPpit_test | eg          | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
```

1. The program to start the database interaction is 'TheSNPpit' just like the name of the whole package. For some this is inconvenient to type. Therefore, a link is created to *snppit*. From now on, the package can also be started through typing *snppit* at the command line.

2. To test if the program can be reached, typing '*snppit -v*' should produce some output along with a version number similar to line 12 from listing 59. If you do not get this but something like '*command unknown*', then you have a mistake in your search path.

3. next we try if you can establish connection with PostgreSQL, using '*psql -l*' as shown in line 15 in listing 59. Again, you should get some output as shown in lines 10 through 15.

4. at this stage you should be able to use TheSNPpit. If there are issues, do not hesitate to get in touch at *eildert.groeneveld@gmx.de* .

6.4 Moving the database

In the default configuration the database itself is located in what is generally, under Linux, considered the root partition, more specifically in */var/lib/PostgreSQL/* . This might not be the brightest place for a large database, as the space of the root partition is generally limited. Also, for backup purposes a different location may be preferred. So it may make sense to talk to your system administrator about this issue.

If you have taken the decision to place the data from PostgreSQL elsewhere you can do this easily by modifying one line in the configuration file *PostgreSQL.conf* which may

be located in "*/etc/PostgreSQL/9.4/main*", the same location as the *pg_hba.conf* that we have already dealt with. In *PostgreSQL.conf* you find the line "*data_directory = '/var/lib/PostgreSQL/9.4/main'* ". to change the location, you have to enter it here, maybe like: "*data_directory = '/data/PostgreSQL/9.4/main'* ". To make this effective PostgreSQL needs to be restarted: "*service PostgreSQL restart*" from root. But before you can do that you need to move the database as described in Listing 60.

.

Listing 60: New location of database

```
1
2  # 1. login as root
3  # 2. go to the new location new location, say: /data:
4  cd /data
5  # 3. stop PostgreSQL
6  service PostgreSQL stop
7  # 4. modify PostgreSQL.conf (see above)
8  # 5. copy the complete PostgreSQL tree
9  rsync -av /var/lib/PostgreSQL .
10 # 6. restart PostgreSQL
11 service PostgreSQL restart
```

This procedure can be done with an empty database or one that has data in it.

The default port for PostgreSQL is *5432*. If you install a new PostgreSQL version parallel to the current version, the new Postgres will be configured with the port *5433*. As TheSNPpit is configured to access the default port *5432*, access will abort. There are two ways to avoid it: firstly, remove the current postgres and then install the new version. Secondly, install the new version in parallel to the current and then modify the port in the line "*port = 5433 # (change requires restart)*", in file "*/etc/PostgreSQL/9.n/main/PostgreSQL.conf*", replacing *5433* by *5432*, and do (all of this as the root user) "*service PostgreSQL restart*".

6.5 Testing the installation

After the installation is complete, a new user may find it useful, to execute predefined commands to get a feel on how TheSNPpit operates. Go back to the section "*Running the demo*" 6.2.3.

6.5.1 Exporting SNP data

In listing 61 we have listed five exports of variable panel sizes.

1. here we have a rather small data set from a 55K panel from 4223 individuals. The export took 9 seconds resulting in a ped file of 890MB.

2. the second export creates a ped file from 500 animals on a high density panel with 835K. This export took also 9 seconds resulting in a 1.6GB file. The extraction speed per second is faster than before at 170 million SNPs because of reduced overhead.

3. number three exports 700 samples from a bigger panel with 2.4 million SNPs resulting in 1.7 billion SNPs being exported. The resultant ped file is a hefty 6.4 GB in size. The export time was 34sec.

4. The final example is an attempt to get a feeling to what level the design of TheSNPpit scales, i.e. how big problems it is able to handle. Here, 150 individuals were exported from the 22 million panel. This amounts to the number of SNPs being reported for the cattle genome, which would certainly be the upper limit for SNP storage requirement in cattle genomic data. 3.3 billion SNPs were exported to a file that grew to 13GB with an export time of one and a half minute. In comparison, exporting to a spinning hard disk reduced the export speed from 83.7 to 50. million SNPs per second.

5. We just pushed TheSNPpit to a panel of 50mio. Also this export of 10 samples went through smoothly indicating that limits as far as panel size and amount of SNPs handled have not been reached.

6. Finally, we loaded 200 more samples from the 50mio panel. This had a files size of 38GB! Exporting now the total of 210 individuals amounted to 10.5 billion! SNPs and went trough in 5 and a half minute. Not bad.

All export were done to a SSD disk. The timings for the number of SNPs/sec were done in the export phase, after the SNP maps had been retrieved, and, thus, reflect the pure data extraction through PostgreSQL from the database, the decoding and recoding in ASCII format as required for the PLINK ped files.

For further information on the scaling of TheSNPpit read our paper in PLOSone [2] .

6.5.2 Storage requirements

Listing 61 shows in line 23 a total storage requirement of around 17GB. The largest storage block is allocated for the SNP description. This is due to the fact that we have huge panels totaling around 75mio SNP and their individual description, which are well beyond anything reasonable these days. This required 9GB of storage (line 29). It should be noted, that storage for the SNP description is independent of the number of genotypes stored.

Listing 61 gives the storage requirement for all 17billion (line 23) genotypes as a mere 4.4GB (line 30). Space for storing the SNP names has to be added which is substantial (9.2GB in this case). However, A SNP name is only stored once, and is therefore the same for 1 sample or 100000. Put that into relation of €200 hard disks of 1 or 2 Terrabyte!

Listing 61: Exporting SNP data:

```
# 1. export 4223 samples from 55K panel: snppit -T -E gs_053
Export finished, written to file gs_053.ss_026.is_053.ped (890MB)
----number of sample_names processed  :  4223
----total number of genotypes written :   233.114 million
----Number of SNP processed per second:    62.01 million
----total execution time: real 0m9.133s

# 2. export 500 samples from 855K panel: snppit -T -E gs_008
Export finished, written to file gs_008.ss_004.is_008.ped (1.6GB)
----number of sample_names processed : 500
----total number of genotypes written:   427.601 million
----Number of SNP processed per second: 170.15 million
----total execution time: real 0m9.157s

# 3. export 700 samples from the 2440K panel: snppit -T -E gs_010
Export finished, written to file gs_010.ss_005.is_010.ped (6.4GB)
----number of sample_names processed : 700
----total number of genotypes written:     1.715 billion
----Number of SNP processed per second:  114.84 million
----total execution time: real 0m24.283s

# 4. export 150 samples from 22Mio panel: snppit -T -E gs_049
Export finished, written to file gs_049.ss_024.is_049.ped (13GB)
----number of sample_names processed : 150
----total number of genotypes written:   3.300 billion
----Number of SNP processed per second: 83.70 million
----total execution time: real 1m26.594s

# 5. export 10 samples from 50Mio panel: snppit -T -E gs_046
Export finished, written to file gs_046.ss_023.is_046.ped (1.9GB)
----number of sample_names processed :   10
----total number of genotypes written:   500 million
----Number of SNP processed per second: 67.78 million
----total execution time: real 1m43.790s

# 6. export 210 samples from 50Mio panel: snppit -T -E gs_055 (38GB)
Export finished, written to file gs_055.ss_027.is_055.ped (40GB)
----number of sample_names processed :   210
----total number of genotypes written:   10.500 billion
----Number of SNP processed per second: 48.01 million
----total execution time: real 5m33.926s
```

6.5.3 Configuration

Some parameters can be configured in the file *TheSNPpit.conf* as shown in Listing 62.

Listing 62: TheSNPpit.conf

```
1  [Default]
2  db_name = TheSNPpit
3  db_password = snpadmin
4  db_host = localhost
5  pagewidth = 200
6
7  [Testing]
8  db_name = TheSNPpit_test
9  db_password = snp_test
10 db_host = localhost
11
12 [Regression]
13 db_name = regression
14 db_password = snp_regression
15 db_host = localhost
16
17 [Export]
18 # available formats: ped, bed, 0125, IB705:
19 default_format = ped
20
21 [Import]
22 # available formats: ped, 0125, IB705:
23 default_format = ped
```

The *[default]* section sets just that: the defaults. Here are some labels the user might want to adapt:

Default: db_name sets the default database. The full snppit calling syntax is:
snppit -T <db_name>
The current default setting is the same as
snppit -T TheSNPpit
being the default the '*-T TheSNPpit*' can simply be dropped and the databases TheSNPpit be accessed.

pagewidth this is the number of characters before a line of output from '*--report*' breaks.

Testing: db_name: specifies the database that is access when only *-T* is specified in the command line, as the name same: useful for testing purposes.

Export: default_format: if no format is specified for export the default is used. In Listing 62 this would be the PLINK *ped* format. If you are using mostly the *0125* format you would replace *ped* by *0125* and would not need to specify format on export

Import: default_format: what has been said about *--export* above applies equally to *--import*. You might receive all of your data in *0125*, then specify it here.

7 Examples

In the following we shall give a few examples exemplifying all steps required from the creation of a new panel, the repeated loading of SNP data, the quality control step, and the export of the SNP set for genomic centered genetic evaluation.

7.1 Loading SNP data

After a new panel has been developed SNP data for the first batch of genotype animals come rolling in. Processing requires two steps. Firstly, the panel needs to be inserted into the database. This contains information which SNPs are part of the panel and where they are located. Let us assume that this is a new high density panel for sheep with 780023 SNPs for which chose the name *sheep_780K*. In the second step the genotype data for the first batch on animals is to be loaded. These have to be in PLINK format consisting of a map and a ped file. Let us assume that the filenames are "*sheep_2014-12wk.ped*" and "*sheep_2014-12wk.map*".

Listing 63: Loading a new panel and first SNP data

```
1 eg(eno,~)snppit --import panel -p sheep_780K -i sheep_2014-12wk.map -f ped
2 eg(eno,~)snppit --import data  -p sheep_780K -i sheep_2014-12wk.ped -f ped
```

In the following weeks new SNP data will accrue and need to be loaded, Let us assume that the corresponding file names are "*sheep_2014-13wk.ped*", "*sheep_2014-14wk.ped*", and "*sheep_2014-15wk.ped*" along with their map files, all of which you should get from your genotyping service. The commands for loading are given in the following block 64.

Listing 64: Loading more SNP data

```
1 eg(eno,~)snppit --import panel -p sheep_780K -i sheep_2014-12wk.map -f ped
2 eg(eno,~)snppit --append panel --name sheep_780K -i sheep_2014-13wk.ped -f ped
3 eg(eno,~)snppit --append panel --name sheep_780K -i sheep_2014-14wk.ped -f ped
4 eg(eno,~)snppit --append panel --name sheep_780K -i sheep_2014-15wk.ped -f ped
```

A bunch of genotype sets is created, one for each *append* and one containing all thus far loaded samples.

7.2 Creating a SNP selection

The full SNP selection is defined through the panel. Thus, a 56K panel has 56K different SNP . Upon loading the first SNP selection list is create as a full list containing all SNPs in the panel. Now, the situation will arise, that only a subset is wanted in an analysis. This could for instance be some SNPs that research has implied in certain phenotypes, and these SNPs are known by name. In this situation, the user can simply create a text file writing on each line one SNP name, where these need to be from the panel definition map. If manual work is involved, this file will likely contain only a few lines. But, of

course, it can basically contain any number of SNP name with the maximum being the panel size. The order of the SNP names do not matter.

Creating a new SNP selection vector is easy, assuming the name of the text file is *snp_restr_1.txt* and the panel *dog_56K*, then this is simply done as shown in listing 65.

Listing 65: Creating a SNP selection based on file input

```
1 snppit -p dog_56K -C snp_selection -i snp_restr_1.txt
2 --> snp_selection created ss_123
```

The output as indicated by the right arrow tells us that a new SNP selection vector has been created, the *ss_123*.

This procedure is the most basic way to create a SNP selection vector. It does not matter how the list of SNP name is created, it will always easily lead to a SNP selection vector. So whenever you can get hold of SNPs and want to create a genotype based on them, this procedure can be used.

Sometimes a selection is more easily defined by what should NOT be included in a SNP selection set. For instance, later information might reveal that SNP *RS_0128981* and *RS_0128235* are actually useless and should therefore never be used. Instead of specifying a SNP selection through a text file containing all SNPs except those two, its simpler to put those two into a file and then use the *-x* flag meaning 'use all SNP from the panel except those in the file *except2.txt*. The command is given in listing 66.

Listing 66: Creating a SNP selection excluding some SNP

```
1 snppit -p dog_56K -C snp_selection -x -i except2.txt
2 --> snp_selection created ss_124
```

7.3 Some editing

Prior to using a genomic dataset, some initial quality control steps are usually employed. The listing 67 shows such an example. In line 1 bad calls are removed from the *gs_004* data set. More specifically, individuals with bad calls of 2% and more will be excluded. In the following steps bad SNPs are controlled, in this example 3% and more are removed. The *--first* indicates if individuals or SNP are controlled first. The resulting genotype set with probably less individuals and SNPs has the name *gs_042*.

Listing 67: Some editing

```
1
2  # 1. restrict no calls to 2 and 3%:
3  snppit -S genotype_set --name=gs_004 --first='ind' --ncind=.02 --ncsnp=.03
4  -->  new genotype_set gs_042
5  # 2. create a new genotype set based on gs_042 with a
6  #    major allele frequency above .03
7  snppit -S genotype_set --name= gs_042 --maf .03
8  -->  new genotype_set gs_043
9  # 3. export this edited genotype set:
10 snppit --export genotype_set --name= gs_043
11 --> exported gs_043.ss_012.is_008.map
```

Notice, that the new genotype set is not actually produced in terms of data volume. Instead, only a new individual and SNP selection vector is created which together form the new genotype set name *gs_042*.

In the next step all SNPs with very low frequency below 3% are removed as specified in line 6. This results in a new genotype set with the name *gs_043.ss_012.is_008.ped* . This file is potentially very large, can be giga bytes.

The export speed will be in the order of 30 – 100 Mio SNPs / second, depending on your computer.

7.4 Creating an individual list and updating individuals

Assume, that you have the notion that one of the individual in the individual selection *is_002* is incorrect. To get all individuals in the selection vector the command in line 2 in Listing 68 is executed. The user notes right away, that there is something wrong with *13987_IIIxx* and *14941_IIIxx*. After making a few phone calls, she finds out that the correct IDs are *13987_Ia* and 1*4941_Id*.

Listing 68: Creating a list of sample names

```
1
2  # 1. create a list of individual in the individual selection is_002
3  snppit -E individual_selection --name is_002
4  --> List of Individuals from is_003
5  --> sample_name
6  --> -----------
7  -->   10730_0
8  -->   12374_0
9  -->   13987_IIIxx
10 -->   14658_0
11 -->   14941_IIIxx
12 -->   15748_0
13 # 2. create an update file upd.txt
14 13987_IIIxx 13987_Ia
15 14941_IIIxx 14941_Id
16 # 3. update sample IDs:
17 snppit --update sampleid -i upd.txt
18 --> number of sample_name updated: 2
```

To be able to make such an update, we create a file (*upd.txt*) that has two lines (as shown in line 13 and 14 in listing 68.

Next we simply run the update function (line 16). We can see from the response that two records have been updated. Notice, that sample IDs in TheSNPpit are unique, that is why on update no reference has to be made to a panel or an individual selection vector.

7.5 Adding G to the BLUP workflow using TheSNPpit

Here we shall describe the steps required to add the data handling for routine BLUP run using genomic data, assuming that this is one such evaluation of a series of many. This could for instance be the weekly BLUP run as is common in pig breeding or the 6 monthly genetic evaluation in cattle. We assume that the pipeline has been set up for such a task, like extracting performance data from the production database, the animals for which the BLUPs are to be computed along with their pedigrees. Clearly, this requires a fully fledged database with consistent pedigree and performance data. It is also assumed that this database can serve as a basis for creating lists of animals to be included in the genetic evaluation with and without SNP data. The steps are:

1. process new SNP data up to the current data and insert in the SNP database
2. extract performance data from database
3. extract pedigree data from database
4. extract list of pertinent animals in the pedigree with SNP data
5. extract genotype records from the SNP database
6. compute relationship matrix on SNP basis
7. compute BLUP on the basis of 2, 3 and 6

Clearly, steps 2, 3, and 7 (without the SNPs) constitute the standard genetic evaluation to which we shall have to add 1 and 5. Let us assume a few things:

1. we are using a 700K panel, which we have called PIG_Il_700K in TheSNPpit and which has already been loaded before
2. we have received 3 new genotype data files with the file names: *PIG_Il_700K_022.ped, PIG_Il_700K_021.ped, PIG_Il_700K_023.ped, PIG_Il_700K_024.ped* and their corresponding map files: *PIG_Il_700K_021.map, PIG_Il_700K_022.map, PIG_Il_700K_023.map*, and *PIG_Il_700K_024.map*. The user has made an arrangement with the genotyping service to have new data exported in PLINK format, this means for each shipment of new genotype data, the customer will receive one ped file and one map file, plus likely some raw data. The latter the user will probably want to stow away safely. Further, the user will keep track of incoming data and rename the file according to her own system like given above and put them in one directory.

3. previous work has resulted in a subset of SNPs being chosen which are to be used in the SNP genetic evaluation. The SNP selection vector has the name *ss_072*.

4. we want to extract SNP data for animals in the file *SNPanimals.txt*. The files has been created by the user as indicated in the above bullet point 4.

5. we want to extract genotypes for the animals in file *SNPanimals.txt* for the SNPs in *ss_072* with the file_name *PIG_Il_700K_wk24.ped* and P*IG_Il_700K_wk24.map* .

The command are issued from the directory, where the input files *PIG_Il_700K_023.ped* etc are located. They can either be started manually, but preferable in a bash script, that contains just these commands as shown in listing 69.

Listing 69: TheSNPpit commands for creating GBLUP files

```
1  # import last genotyping
2  eg(eno,~)snppit -A PIG_Il_700K -i PIG_Il_700K_021.ped -f ped
3  eg(eno,~)snppit -A PIG_Il_700K -i PIG_Il_700K_022.ped -f ped
4  eg(eno,~)snppit -A PIG_Il_700K -i PIG_Il_700K_023.ped -f ped
5  eg(eno,~)snppit -A PIG_Il_700K -i PIG_Il_700K_024.ped -f ped
6
7  # output from last snppit run: all animals and all SNP in gs_086
8  --> gs_086|is_098|ss_008|2014-09-27 11:19:31 All IDs from PIG_Il_700K_024.ped
9
10 # create individual selection for animals in SNPanimals.txt
11 snppit -p PIG_Il_700K -C individual_selection -i SNPanimals.txt
12 --> individual_selection is_099 created
13
14 #create a genotype_set for export:
15 snppit -C genotype_set -n ss_076 -s is_087 -c 'GS for export from week 24'
16 --> gs_087|is_099|ss_076|2014-09-27 11:19:31 GS for export from week 24
17
18 # export the new genotype set:
19 snppit -E gs_087 -o PIG_Il_700K_wk24
20 --> Export finished, written to file      PIG_Il_700K_wk24.ped
21 --> Number of SNP processed per second:    35.75 million
```

Lines 2–5 are the commands to load or import the new SNP data for the panel *PIG_Il_700k*, each producing a few lines of outputs. The output relevant to the following step is shown in line 8, which states that the genotype set *gs_086* contains all SNP data on all animals. But this is of course not the dataset that is necessarily wanted for GBLUP. Firstly, not all individuals may be required, secondly, not all SNPs will be used. Thus, an reduced set, a subset, will be required. Genotype sets are defined through two lists: the individual selection list, and the SNP selection list. As stated above, the GBLUP workflow assumes, that the file *SNPanimals.txt* contains the IDs of animals for which genotype data are to be used in the genetic evaluation. This is simply a text file that contains all IDs, one per row.

Line 11 in listing 69 reads the file *SNPanimals.txt* and creates an individual selection vector. The output (shown in line 12) gives us the name as *is_099*. Next we require a SNP selection vector for the definition of a genotype set (which we finally want to export

for input to GBLUP). As stated above we assume that earlier research has defined a set of SNPs that were found to be appropriate to define the SNP set to be used for GBLUP. This has been created in analogy to line 11 and has resulted in an SNP selection with the name of *ss_072*.

Line 15 in listing 69 now creates the new genotype_set on the basis of the content of *SNPanimals.txt* and the *ss_079* with the name *gs_087*. Now one last step is required to export the SNP genotypes required for GBLUP input. This step is executed as shown in line 19.

8 Scope

TheSNPpit has been developed to handle massive amounts of SNP data. To that effect it uses highly compressed data storage and a novel storage mechanism together with highly optimized C code for imports and exports. TheSNPpit scales basically linearly with the number of SNPs. For a detailed discussion see Groeneveld and Lichtenberg[2]. TheSNPpit has been run to import massive amounts of data: database size for more than 18 million samples have been loaded thus far with 3.4 Trillion genotypes as shown in Listing 70.

Listing 70: TheSNPpit scaling

```
1
2  List of SNP panels
3
4   Panel   |  nSNP   |nSample  |SNP(mio)
5  ---------|---------|---------|--------
6  P.01000K|1000000  |1000800  |1000800
7  P.00200K| 200000  |4525000  | 905000
8  P.00100K| 100000  |5533000  | 553300
9  P.00700K| 700000  | 525000  | 367500
10 P.00054K| 54000   |5525899  | 298398
11 P.00500K| 500000  | 526600  | 263300
12 P.20000K|20000000|    40    |  800
13 P.00001K|  1000   | 800000  |  800
14 P.05000K|5000000  |  160    |  800
15 P.10000K|10000000|    80    |  800
16 ..
17  Total   |   -     |18526843|3393650
18
19             Database size
20
21           Tables             |total_size
22 ----------------------------|----------
23 public.genotype_data         |  840 GB
24 public.snp                   | 5095 MB
25 public.individual_selection| 3359 MB
26 public.individual            | 1023 MB
```

Export speeds vary between 50 and 250mio SNP per second depending on computer, disk type (SSD much faster than hard disk) and output format.

9 Contributed/Loading Affimetrix Data

As has been described, TheSNPpit imports natively PLINK *ped* and *0125* formatted data. From the initial phase of TheSNPpit development a set of programs have been developed that produce PLINK *ped/map* files from the standard Affimetrix data. These are bash scripts using awk and a Fortran90 program *plate2plink* for producing the final ped files. Interested uses can send an email to obtain the programs. The description below allows the user to assess, if our setup is applicable to their own.

The following data is supplied:

1. the current annotation file which may get superseded by later versions:

   a) Axiom_GW_GT_chk.na32.annot.csv

```
#%create_date=2012-12-20 GMT-08:
#%chip_type=Axiom_GW_GT_Chicken
#%genome-species=Gallus gallus
#%genome-version-ncbi=Gallus_gall
#%genome-version-create_date=2011
#%netaffx-annotation-date=2012-11-05
#%netaffx-annotation-netaffx-build=33
#%netaffx-annotation-tabular-format-v
"Probe Set ID","Affy SNP ID","dbSNP
"AX-75183725","Affx-79509805","---",
"AX-75183727","Affx-79509925","---",
"AX-75183728","Affx-79511500","---",
"AX-75183731","Affx-79511118","---",
"AX-75183733","Affx-79510989","---",
"AX-75183734","Affx-79510060","---",
```

   The first column holds the SNP labels given and used by Affimetrix throughout their data: they are to be used in the genotype data file.

   b) for each Plate on directory. Here, for this import only *AxiomGT1.calls.txt* are used, which contains the SNP data.

```
#Calls: -1=NN, 0=AA, 1=AB, 2=BB
#%guid=5a93b615-1aa8-4241-8091-5023cfaaf9d6
probeset_id 1231_F04.CEL 1739_E03.CEL 2174_H07.CEL
AX-75183725 2 2 2 1 2 2 2 2 2 0 2 2 2 2
AX-75183727 1 1 1 1 1 2 2 2 2 2 2 1 2 2
AX-75183728 0 2 -1 0 0 0 0 0 0 0 0 0 1 0
AX-75183731 0 2 2 2 2 1 2 2 2 1 2 2 2 2
AX-75183733 0 0 0 0 0 0 1 0 0 0 0 0 0 0
AX-75183734 2 0 0 0 0 0 0 0 0 0 1 1 1 0
AX-75183735 2 0 2 2 1 2 1 2 2 1 2 2 1 1
AX-75183738 2 2 2 2 1 2 2 2 0 2 2 2 2 2
```

   Thus, there is one row per SNP with the sample ID going across. This mean, that the genotypes are fully defined in the file *AxiomGT1.calls.txt* . Chromosome position information is located in the annotation file. From what we have seen, the order of the SNP in the two files are identical, however, this must get checked

in loading. If an error is made here, nobody will find out, only the results will the wrong (a nightmare should this happen). Meanwhile, this check has been implemented in the *create_PLINK_file-f90* script.

Loading of these SNP data into TheSNPpit is done in two steps:

1. a PLINK compatible set of input file is created for TheSNPpit to load. These are the *PLINK.map* and *PLINK.ped* files.
   a) This is done with the bash script "*create_PLINK_data-f90*" using for instance "*create_PLINK_file-f90 -a Axiom_GW.na33.annot.csv -m newPLINK*". In the process, beginning from the current directory, all subdirectories are scanned for files '**calls.txt*' which are then processed one after the other. The annotation file needs to be present as given in the bash call. Its purpose is two fold. Firstly, the *PLINK.map* file is created from it, assuming that the annotation has not changed for the files in the directory tree. Secondly, for each plate the sequence in the plate and in the map file is checked for identity. The program aborts if the order is not the same. Then the user will have to investigate the reason for this out-of-sync.
   b) the bash script is a wrapper around the the Fortran90 program *plate2PLINK* which is called with the four parameters nsamples, nsnp, and infile outfile. *nsamples* is required because not always all 96 wells are good. This information is extracted from the **calls.txt* file by the "*create_PLINK_data-f90*" script.
2. The panel is loaded from the "*ped.map*"
3. The genotype data are loaded from the *PLINK.ped* file.

Thus, the complete sequence of program invocation is:

Listing 71: Software installation

```
eg(eno,PLATES):ls
Axiom-na33.annot.csv  newPLINK.map  newPLINK.ped  PLATE1  PLATE2
eg(eno,eg):create_PLINK_file-f90 -a Axiom-na33.annot.csv
  -m 2plates -o 2plates
PLINK formatted map file has been created : 2plates.map
 580961 SNP will be loaded in this run from 2 plates
Plate 1 done with  96 good samples Mo 27. Jan 19:35:23 CET 2014
Plate 2 done with  96 good samples Mo 27. Jan 19:35:32 CET 2014
------------------------------------------------------------
- 192 animals have been converted from 2 plates
- In total 111 mio genotypes have been written to 2plates.ped
------------------------------------------------------------
real    0m21.229s user  0m17.179s sys   0m3.222s

eg(eno,PLATES):ls
Axiom-na33.annot.csv  2plates.map  2plates.ped  PLATE1  PLATE2
eg(eno,2Plates-dir): snppit -I panel -p chick600-33  -i 2plates.map  -f ped
2014-02-03 12:21:26,505 eno SNPLib.pm(87) INFO: Panel chick600-33 inserted
2014-02-03 12:21:26,505 eno SNPLib.pm(91) INFO: Retrieving snp names ...
2014-02-03 12:21:29,370 eno PLINK.pm(59) INFO: Read 580961 SNP names along
                                              with chromosome information
2014-02-03 12:21:29,400 eno SNPLib.pm(104) INFO: Writing snp names into
                                                 the database ...
2014-02-03 12:21:40,734 eno SNPLib.pm(109) INFO: 580961 SNPs inserted
2014-02-03 12:21:40,736 eno SNPLib.pm(114) INFO: Panel insert committed
eg(eno,2Plates-dir):snppit -I data -p chick600-33  -i 2plates.ped  -f ped
     ...
     ...
    192 samples imported in database.
```

10 Acknowledgements

The proof of concept version was programmed by Truong Van Chi Cong and Helmut Lichtenberg. Most of that structure and Perl code is still in operation. Further, I have to admit, that Helmut got regularly victimized as he had to listen to and discuss my questions/ideas/worries, and it was always very useful (for me and definitely also the project).

11 Citation

For a citation please use the publication in PLOS ONE: [2]

References

[1] Christine Fong, Dennis C. Ko, Michael Wasnick, Matthew Radey, Samuel I. Miller, and Mitchell Brittnacher. Original paper - GWAS analyzer: integrating genotype, phenotype and public annotation data for genome-wide association study analysis. *Bioinformatics*, 26(4):560–564, 2010. doi:10.1093/bioinformatics/btp714.

[2] E Groeneveld and H Lichtenberg. TheSNPpit – a high performance database system for managing large scale SNP data. *PLOS ONE*, 2016. doi: 10.1371/journal.pone.0164043.

[3] Eildert Groeneveld and Cong VC Truong. A database for efficient storage and management of multi panel SNP data. *Archiv Tierzucht / Archives Animal Breeding*, 56(103), 2013. ISSN:0003-9438; doi:10.7482/0003-9438-56-103.

[4] Faheem Mitha, Herodotos Herodotou, Nedyalko Borisov, Chen Jiang, Josh Yoder, and Kouros Owzar. SNPpy - Database Management for SNP Data from GWAS Studies. In *Duke Biostatistics and Bioinformatics (B&B)*, number paper 14 in Working Paper Series, page 19 pp. Duke University, 2011. http://biostats.bepress.com/dukebiostat/papers/art14.

[5] Benjamin Neale, Kathe Todd-Brown, Lori Thomas, Manuel A R Ferreira, David Bender Julian Maller, Paul I W de Bakker, Mark J Daly, and Pak C Sham. PLINK: a toolset for whole genome association and population-based linkage analyses. http://pngu.mgh.harvard.edu/purcell/plink/. Correspondence: Shaun Purcell, Rm 6.254, CPZ-N, 185 Cambridge Street, Boston, MA, 02114, USA; Tel: 617-726-7642; Fax: 617-726-0830; shaun@pngu.mgh.harvard.edu.

[6] A. Orro, G. Guffanti, E. Salvi, F. Macciardi, and L. Milanesi. Snplims: a data management system for genome wide association studies. *BMC bioinformatics*, 9 Suppl 2, 2008.

Index