TASSEL 3.0 Genotyping by Sequencing (GBS) pipeline documentation

Authors: Jeff Glaubitz, James Harriman, Terry Casstevens

Acknowledgements: Many thanks to Genevieve DeClerk and Charles Chen for excellent suggestions for improving this documentation. They are not to be blamed for its many remaining deficiencies, however.

Please note that this is an unfinished work in progress…

Table of Contents

Introduction

The GBS analysis pipeline is an extension to the Java program TASSEL, and, as such, GBS commands are run as TASSEL plugins via the command line in the following format (Linux or Mac operating system; for Windows use `run_pipeline.bat`):

```
   run_pipeline.pl -fork1 -PluginName --plugin-option -endPlugin -runfork1
```

Each step of the pipeline is specified with a "fork" command and a number, since TASSEL can run several processes at once, and split and recombine their results. The fork option is followed by the name of the plugin, and any plugin-specific options. If no plugin options are provided, the program will print a list of available options. -endPlugin signals the end of plugin-specific options, and -runfork1 then runs the specified plugin. In all of our examples here for the GBS pipeline, we run only a single fork at a time.

Please see http://www.maizegenetics.net/tassel/docs/TasselPipelineCLI.pdf for general instructions on how to install the TASSEL 3.0 Standalone Build on your computer. These GBS-specific instructions assume that you have unzipped the standalone into the directory (folder)

```
      /programs
```

and then renamed the directory

```
      /programs/tassel3.0_standalone
```

to

```
      /programs/tassel
```

If not, you will have to edit the example commands appropriately (*e.g.*, replace "`tassel`" with "`tassel3.0_standalone`").

If you have more memory available on your machine than 1.5GB, then you can increase the amount of memory available to TASSEL by opening `run_pipeline.pl` (or `run_pipeline.bat`) in a text editor and modifying "`-Xms512m -Xmx1536m`" to (for example) "`-Xms512m –Xmx4096m`" (the `–Xmx` option controls the maximum memory allocation).

Note that many of the GBS commands ("plugins") produce a large amount of (poorly formatted) console output ("stdout") that is not discussed below (at least not in this incomplete draft). Much of this output is very informative, so you will likely find it helpful to either copy and paste it to a text log file or redirect stdout to both the console and a log file (*e.g.*, using '`| tee GBSlogfile20110915.txt`' in Linux). Alternatively, if you are working in a Linux Bash shell you can start a new shell with the command '`bash | tee -a logfile.txt`'. All of your keystrokes along with the console output to this new shell should then be appended (`-a` option) to the logfile.txt (rename the file as you see fit).

The next version of this documentation will include a flow chart showing how the steps of the analysis link together.

A more concise, less detailed example of running the pipeline is available in this tutorial: http://cbsu.tc.cornell.edu/lab/doc/gbs_pipeline_workshop_walkthrough.pdf.

Recommended directory (folder) structure for a GBS analysis

A dot (.) will represent the working directory (folder) for your analysis, which will be your current working directory (e.g., `/home/myUserName/myGBSstudyName`)

The example commands below don't create the directories (and will fail if they don't already exist), so at the start of the analysis, create the following directories inside your working directory.

```
./qseq OR ./fastq   (original raw data files, one file per flowcell lane)
./tagCounts  (for output from QseqToTagCountPlugin OR FastqToTagCountPlugin)
./topm              (for output from SAMConverterPlugin)
./mergedTagCounts   (for output from MergeMultipleTagCountPlugin)
./tbt               (for output from QseqToTBTPlugin OR FastqToTBTPlugin)
./mergedTBT         (for output from MergeTagsByTaxaFilesPlugin)
./hapmap
./hapmap/unfilt     (for output from TagsToSNPByAlignmentMTPlugin)
./hapmap/mergedSNPs (for output from MergeDuplicateSNPsPlugin)
./hapmap/filt       (for output from GBSHapMapFiltersPlugin)
```

QseqToTagCountPlugin

Summary:
Derives a tagCount list for each qseq file in the input directory (and all subdirectories thereof). Keeps only good reads having a barcode and a cut site and no N's in the useful part of the sequence. Trims off the barcodes and truncates sequences that (1) have a second cut site, or (2) read into the common adapter. **If your input files are in fastq format (and qseq files are not available), use FastqToTagCountPlugin instead (same arguments)**.

Input:
- Barcode key file (see example in Appendix 1)

- Directory (folder) containing qseq files

Output:
- Directory (folder) containing a corresponding tagCount (*.cnt) file for every qseq file in the input directory

Arguments:

QseqToTagCountPlugin	
-i	Input directory containing .qseq text or gzipped text files. NOTE: Directory will be searched recursively, and should be written without a slash after its name.
-k	Key file listing barcodes for each sample.
-e	Enzyme used to create the GBS library (ApeKI or PstI).
-s	Maximum number of good reads per lane. Default: 200,000,000
-c	Minimum number of times a tag must be present to be output. Default: 1
-o	Output directory to contain .cnt files, one per .qseq file. Defaults to input directory (the default is not recommended – it is best to use a separate directory).

Example command:
```
/programs/tassel/run_pipeline.pl -fork1 -QseqToTagCountPlugin -i qseq
-k myGBSkey.txt -e ApeKI –o tagCounts -endPlugin -runfork1
```

Gory Details:
This is the initial step of a GBS analysis. This step reads a user-supplied key file (mandatory argument **-k**) in tab-delimited text format which indicates, for each lane of interest from a flowcell, which barcodes are assigned to which sample (a short example key file is provided in Appendix 1). It then recursively searches the specified input directory (mandatory argument **-i**) and all subfolders for qseq files matching one of the flowcell/lane combinations in the key file and with the following acceptable file naming conventions:

FLOWCELL_LANE_qseq.txt	(example: **42A87AAXX_2_qseq.txt**)
FLOWCELL_LANE_qseq.txt.gz	(example: **42A87AAXX_2_qseq.txt.gz**)
code_FLOWCELL_s_LANE_qseq.txt	(example: **10225395_42A87AAXX_s_2_qseq.txt**)
code_FLOWCELL_s_LANE_qseq.txt.gz	(example: **10225395_42A87AAXX_s_2_qseq.txt.gz**)

Note that both compressed (*.gz) and uncompressed (*.txt) files can be read – we recommend using compressed files to save disk storage space. The “code” part of the latter two file name examples is a numerical tracking code generated by our sequencing center. Our GBS pipeline doesn’t actually use this code, so you can substitute any text or numbers (or use one of the first two conventions). The underscores are essential for correct parsing of the parts of each qseq file name (only FLOWCELL and LANE are actually used by our pipeline).

For each qseq file that has a match in the key file, QseqToTagCountPlugin finds all reads that begin with one of the expected barcodes immediately followed by the expected cut site remnant (CAGC or CTGC for *ApeK*I) and trims them to 64 bases (including the cut site remnant but removing the barcode). Reads containing N within the first 64 bases after the barcode are rejected. If a read contains either a full cut site (from incomplete digestion or chimera formation) or the beginning of the common adapter (from restriction fragments less than 64bp) within the first 64 bases it is truncated appropriately and padded to 64 bases with polyA. The actual length of truncated (or full 64 base) reads is recorded in the output tagCount file.

The output of QseqToTagCountPlugin is a single tagCount file in the specified output directory (mandatory argument **-o**) for every matching qseq file in the input directory. The tagCount files are named after their corresponding qseq file, with *_qseq.txt.gz or *_qseq.txt replaced by *.cnt. The tagCount files are binary, and can only be read by our pipeline. They contain the 64 base sequence of each good, barcoded tag (padded with polyA if truncated), the actual length of the tag (before padding with polyA), and the number of times that tag was observed in the corresponding flowcell lane. The tags are sorted by their sequence.

The enzyme used to create the GBS library is indicated via mandatory option **-e**. Currently, our pipeline only accepts ApeKI or PstI. The **-s** option (*maximum number of good reads per lane*) is used to set an upper limit on memory usage. So far we have not encountered a qseq (or fastq) file with more than 200 million good, barcoded reads (the default).

We usually keep the **-c** option (*minimum number of times a tag must be present to be output*) at its default value of 1. In a typical analysis, we usually combine the results of multiple lanes, or even multiple flow cells, via the next step, MergeMultipleTagCountPlugin. Tags that occur only once in a given flowcell lane might occur multiple times in other lanes, so they might be real (*i.e.*, not from sequencing error).

We recommend using qseq files if you have them because they contain all reads, not just the ones passing Illumina's quality filters. We have found that perfectly good reads (exactly matching a 64 base tag that we have seen many times) can be filtered out by Illumina. **If qseq files are not available, or your raw data are in Illumina's latest FASTQ format (from Casava 1.8), use FastqToTagCountPlugin instead (same arguments as QseqToTagCountPlugin).**

MergeMultipleTagCountPlugin

Summary:
Merges each tagCount file in the input directory into a single "master" tagCount list. Only keeps tags with a total count (after merger) greater than or equal to that specified in option **-c** (*minimum number of times a tag must be present to be output*).

Input:
- Input directory (folder) containing tagCount (*.cnt) files

Output:
- Merged tagCount file (it is best to send this to a separate directory from the input directory)

Arguments:

MergeMultipleTagCountPlugin	
-i	Input directory containing tagCount (*.cnt) files.
-o	Output file name (should be in a separate directory from the input).
-c	Minimum number of times a tag must be present to be output. Default: 1
-t	Specifies that reads should be output in FASTQ text format.

Example command:
```
/programs/tassel/run_pipeline.pl -fork1 -MergeMultipleTagCountPlugin
-i tagCounts –o mergedTagCounts/myMasterTags.cnt –c 5 -endPlugin -runfork1
```

Gory Details:
The MergeMultipleTagCountPlugin step merges multiple tagCount files produced by the QseqToTagCountPlugin step (from multiple lanes and/or flowcells) into a single "master" tagCount file. (For a description of the tagCount file format, see QseqToTagCountPlugin.) All tagCount (*.cnt) files in the specified input directory (argument **-i**) are merged.

To remove rare or singleton tags that possibly result from sequencing errors, we use the **-c** option (*minimum number of times a tag must be present to be output*). A **-c** option setting of 10 is typical, but when deciding on an appropriate cutoff, you should consider the number of individuals in your analysis, the expected coverage (about 0.4-0.5x for maize with *Ape*KI), the expected segregation ratio, minimum minor allele frequency of interest, etc. The merged tagCount output file is used as a master tag list for two subsequent steps: the QseqToTBTPlugin step and alignment to the reference genome. The output is in (binary) tagCount format by default, which serves as the input format for the QseqToTBTPlugin step.

We typically perform the alignment to the reference genome with external software, such as BWA. To obtain a master tagCount file in FASTQ format for use as input to BWA, invoke the **-t** option.

SAMConverterPlugin

Summary:
Converts a SAM format alignment (*.sam) file produced by the Unix program BWA into a tagsOnPhysicalMap (*.topm.bin) file that can be used by the TagsToSNPByAlignmentMTPlugin for calling SNPs.

Input:
- SAM format alignment (*.sam) file produced by the Linux program BWA

Output:
- tagsOnPhysicalMap (*.topm.bin) file that can be used by the TagsToSNPByAlignmentMTPlugin for calling SNPs

Arguments:

SAMConverterPlugin	
-i	Input SAM format alignment (*.sam) file produced by the Unix program BWA.
-o	Output tagsOnPhysicalMap (*.topm.bin) file that can be used by the TagsToSNPByAlignmentMTPlugin for calling SNPs. We recommend using the extension *.topm.bin.

Example command:
```
/programs/tassel/run_pipeline.pl -fork1 -SAMConverterPlugin
-i mergedTagCounts/myAlignedMasterTags.sam
-o topm/myMasterTags.topm.bin -endPlugin -runfork1
```

Gory Details:
The next draft of this documentation will include more information on running BWA using the output of MergeMultipleTagCountPlugin (using -t option) as input to BWA and then using BWA to produce a *.sam file to be used as input to this SAMConverterPlugin. For an example, have a look at steps 5-7 here: http://cbsu.tc.cornell.edu/lab/doc/gbs_pipeline_workshop_walkthrough.pdf. However. these steps assume that the original fasta file containing the reference genome has already been formatted and indexed using BWA – consult the BWA documentation for guidance on this.

One important point is that, for the SAMConverterPlugin to work correctly, the chromosome names in the original fasta file containing the reference genome must be integers, *e.g.*, >1, >2, >3, etc. rather than >chrom1, >chrom2, >chrom3, etc.

QseqToTBTPlugin

Summary:
Generates a TagsByTaxa file for each qseq file in the input directory (or in subfolders thereof). One TagsByTaxa file is produced per qseq file. Requires a master list of tags of interest, which may come either from a tagCount or tagsOnPhysicalMap file. **If your input files are in fastq format (and qseq files are not available), use FastqToTBTPlugin instead (same arguments).**

Input:
- Barcode key file (see example in Appendix 1)
- Directory (folder) containing qseq files

Output:
- Directory (folder) containing a corresponding tagsByTaxa file for every qseq file in the input directory

Arguments:

QseqToTBTPlugin	
-i	Input directory containing .qseq files.
-k	Barcode key file.
-e	Enzyme used to create the GBS library (ApeKI or PstI).
-o	Output directory.
-c	Minimum taxa count within a qseq file for a tag to be output. Default: 1
-t	Tag count file listing unique reads (mutually exclusive with option -m).
-m	Physical map file listing unique reads (mutually exclusive with option -t).
-y	Output in TBTByte format (counts from 0-127) instead of TBTBit (0 or 1)

Example command:
```
/programs/tassel/run_pipeline.pl -fork1 -QseqToTBTPlugin -i qseq
-k myGBSkey.txt -e ApeKI –o tbt –t mergedTagCounts/myMasterTags.cnt
-endPlugin -runfork1
```

Gory Details:
Similar to QseqToTagCountPlugin, QseqToTBTPlugin parses qseq files for good reads that contain a barcode and cut site remnant and that have no N's in the first 64 bases after the barcode, and trims them to 64 bases (not including the barcode). As in QseqToTagCountPlugin, QseqToTBTPlugin appropriately truncates reads that contain either a full cut site or the beginning of the common adapter within the first 64 bases, and pads them to 64 bases with polyA. In a given GBS analysis, the same key file (**-k** option), containing the names of the taxa corresponding to each barcode in each lane, is used for both plugins (see Appendix 1 for an example key file).

The difference between QseqToTBTPlugin and QseqToTagCountPlugin is that QseqToTBTPlugin uses the barcode information to keep track of which taxa each tag of interest is observed in. Each good read in each qseq file is checked for a match against a set of tags of interest. A tagsByTaxa output file is produced for every qseq file in the input directory that has a matching flowcell and lane in the key file. Each output file is named after its corresponding input qseq file but with "_qseq.txt.gz" or "_qseq.txt" replaced by ".tbt.bin". Each output tagsByTaxa file is in binary format (only readable by our pipeline), but can be thought of as a grid where the rows are the tags of interest, the columns are taxa names (including flowcell, lane and well information) and the cells indicate whether or not a particular tag was observed in a particular taxon. The actual length in bases of each tag (not including the polyA padding) is also recorded.

The set of tags of interest are those that are present in the input master tagCount file (using the **-t** option) or tagsOnPhysicalMap file (using the mutually exclusive **-m** option). We usually use the **-t** option, using the output of MergeMultipleTagCountPlugin as the **-t** option input file.

Our pipeline currently supports only the enzymes ApeKI and PstI (**-e** option).

We generally leave the **-c** option (*minimum taxa count within a qseq file for a tag to be output*) at its default value of 1. Filtering of *tags* based upon the number of taxa they appear in would be better performed at the MergeTagsByTaxaFilesPlugin step, but is not currently implemented (however, filtering of *SNPs* based upon data coverage/amount of missing data can be performed with the GBSHapMapFiltersPlugin). With the default **-c** option of 1, tags that are in the master tagCount file but are not present in a given qseq file will not be output into the corresponding tagsByTaxa file – this is a good thing, as it saves disk space (no need to store rows containing nothing but zeros).

If qseq files are not available or your raw data are in Illumina's latest FASTQ format (from Cassava 1.8), use FastqToTBTPlugin instead (same arguments as QseqToTBTPlugin).

The multiple tagsByTaxa files produced by this QseqToTBTPlugin can be merged into a single master tagsByTaxa file in the next step, MergeTagsByTaxaFilesPlugin.

MergeTagsByTaxaFilesPlugin

Summary:
Merges all *.tbt.bin files present in the input directory and all of its subdirectories.

Input:
- Directory (folder) containing multiple tagsByTaxa (*.tbt.bin) files (produced by QseqToTBTPlugin)

Output:
- Merged tagsByTaxa file (it is best to send this to a separate directory from the input directory)

Arguments:

MergeTagsByTaxaFilesPlugin	
-i	Input directory containing multiple tagsByTaxa (*.tbt.bin) files.
-o	Output file name (should be in a separate directory from the input). We recommend using the extension *.tbt.bin.
-s	Maximum number of tags the TBT can hold while merging (default: 200,000,000). Reduce this only if you run out of memory (omit the commas).
-x	Merges tag counts of taxa with identical names. Not performed by default.

Example command:
```
/programs/tassel/run_pipeline.pl -fork1 -MergeTagsByTaxaFilesPlugin
-i tbt –o mergedTBT/myStudy.tbt.bin -endPlugin -runfork1
```

Gory Details:
This step merges the separate tagsByTaxa files produced by the QseqToTBTPlugin (and/or FastqToTBTPlugin) into a single, experiment-wide tagsByTaxa file for all of the flow cell lanes in your experiment. Currently, only the presence or absence of each tag in each taxon is recorded.

The **-s** option controls the maximum number of tags that can be stored in the TBT tag list during the merger process. It defaults to 200,000,000. This is much larger than is needed for most purposes. If you try to run MergeTagsByTaxaFilesPlugin but run out of memory, invoke this option with a number smaller than 200,000,000. Use the largest possible number that your memory capacity can handle. This should be *at least* twice the number of tags in the master tagCounts (or master tagsOnPhysicalMap) file that you used to generate the individual tagsByTaxa files (in the QseqToTBTPlugin).

The **-x** option (off by default) can be invoked to merge the tag counts of taxa with identical names in the key file but from different flow cells, lanes or barcodes within a lane. However, we recommend leaving in any duplicated taxa for now as they can be used in a later step (GBSHapMapFiltersPlugin or MergeIdenticalTaxaPlugin) to check error rates.

TagsToSNPByAlignmentMTPlugin

Summary:
Aligns tags from the same physical location against one another, calls SNPs from each alignment, and then outputs the SNP genotypes to a HapMap format file (one file per chromosome).

Input:
- TagsByTaxa file (*.tbt.bin) indicating the presence or absence of each tag of interest in each taxon
- TagsOnPhysicalMap file (*.topm.bin) containing genomic position of each tag of interest.

Output:
- Directory (folder) containing a HapMap format genotype file (one file per chromosome).

Arguments:

TagsToSNPByAlignmentMTPlugin	
-i	Input TagsByTaxa (*.tbt.bin) file.
-o	Output directory. Defaults to current directory (the default is not recommended – it is best to use a separate directory such as './hapmap/unfilt').
-m	TagsOnPhysicalMap (*.topm.bin) file containing genomic position of tags.
-mnF	Minimum value of F (inbreeding coefficient). Not tested by default.
-mnMAF	Minimum minor allele frequency. Defaults to 0.01. SNPs that pass *either* the specified minimum minor allele frequency (mnMAF) or count (mnMAC) will be output.
-mnMAC	Minimum minor allele count. Defaults to 10. SNPs that pass *either* the specified minimum minor allele count (mnMAC) or frequency (mnMAF) will be output.
-mnLCov	Minimum locus coverage, *i.e.*, the proportion of taxa with at least one tag at the locus. Default: 0.1
-inclRare	Include the rare alleles (3rd or 4th states) at site. These are ignored by default (genotype set to missing).
-inclGaps	Include sites where the major or minor allele is a gap. These sites are ignored by default.
-s	Start chromosome.
-e	End chromosome.

Example command:
```
/programs/tassel/run_pipeline.pl -fork1 -TagsToSNPByAlignmentMTPlugin
-i mergedTBT/myStudy.tbt.bin -m topm/myMasterTags.topm.bin
-o hapmap/unfilt -s 1 -e 10 -mnF 0.9 -endPlugin -runfork1
```

Gory Details:
In this step, a sequence alignment is created for each set of tags that align to the exact same genomic position and strand (where the starting point is defined by the barcode end of the tag) and SNPs are called from each alignment. Tags with multiple or unknown physical genomic positions are not used. The SNP calls from each set of alignments are written to a genotype file in HapMap format, with one HapMap file produced per chromosome. These output HapMap files are named after the input tagsByTaxa file, with ".tbt.bin" replaced with ".c#.hmp.txt", where # is a chromosome number (*e.g.*, *.c1.hmp.txt)

If you are working with highly homozygous inbred lines or a selfing species, then be sure to use the **-mnF** (minimum F) option (we suggest setting mnF to 0.9), where 'F' means 'inbreeding coefficient'. In species like maize which contain abundant paralogs (from ancient chromosomal duplications), this can filter out numerous bad SNPs. **If you are not working with inbred lines or a selfing species, then do not invoke the -mnF option.**

The options **-mnMAF** (minimum minor allele frequency) and **-mnMAC** (minimum minor allele count) can be used to filter out SNPs with rare minor alleles that possibly result from sequencing errors. Keep in mind that SNPs that pass *either* of these criteria will be output, so there is no point in having one of them set stringent but the other too lax. If you are working with a biparental family with 1:1 segregation you might try a mnMAF of 0.2 and a highly stringent mnMAC close to your total number of taxa, so that it is irrelevant (in that case, only the mnMAF will matter). With unrelated individuals and no way to test segregation or LD, you might want to try a mnMAF of 0.02 (and a highly stringent mnMAC close to your total number of taxa).

The **-mnLCov** (minimum locus coverage) option can be used to filter out SNPs with very high amounts of missing data from the output. These most likely result from large restriction fragments (>400 bp) that are not amplified as efficiently in the PCR steps of the GBS protocol. The default value mnLCov of 0.1 should suffice for most purposes.

We recommend that you do not invoke the **-inclRare** option, so that 3rd and 4th allelic states (*i.e.*, triallelic and quadra-allelic SNPs) are ignored (genotypes set to missing). Any 3rd and 4th allelic states are far more likely to result from sequencing error than biological reality.

Similarly, we recommend that you do not invoke the **-inclGaps** option, so that small indels are not scored. Because of alignment issues for small indels (multiple equally likely alignments), they can end up being positioned slightly differently in replicate runs of the plugin. Also, because our tags are all 64 bases (or smaller) in length, small indels in the middle of a tag alignment always result in artifactual, compensatory small indels of equal size at or near the end of the tag alignment. However, if you are interested in maximizing marker saturation (for example, for GWAS or for fine-mapping of a QTL), then you might want to invoke inclGaps: there will almost certainly be numerous sets of tag alignments that contain no SNPs but do contain a small indel. Note that with inclGaps invoked, a three base indel (for example) will be output as three consecutive single base gaps in the HapMap file (plus an additional three artifactual, single base gaps). If the insertion is not present in the reference genome, the three real gaps will all have the same position (the base in the reference genome immediately preceding the insertion). Essentially they are redundant scorings of the same indel.

The HapMap genotype files that we generate save disk space and memory by using single letters to represent phase unknown, diploid genotypes. Heterozygotes are represented by IUPAC nucleotide codes:

```
A = A/A
C = C/C
G = G/G
T = T/T
M = A/C
R = A/G
W = A/T
S = C/G
Y = C/T
K = G/T
N = missing data
```

The “MT” in the name of this plugin indicates that it was initially written to run faster by using multiple threads on multiple CPUs. However, we found that this caused some difficult to trace bugs, so the multiple threading is currently disabled.

Genotypes from tags matching the minus strand of the reference genome are complemented so that they are recorded relative to the plus strand. Hence, all SNPs in the output are relative to the plus strand. For restriction fragment smaller than 128bp, the (plus and minus strand) reads from opposite ends can overlap and assay the same SNPs. Hence, the output of TagsToSNPByAlignmentMTPlugin will contain some duplicate SNPs, each with different patterns of missing data. These duplicate SNPs can be merged in the next step of the analysis, with the MergeDuplicateSNPsPlugin.

MergeDuplicateSNPsPlugin

Summary:

Finds duplicate SNPs in the input HapMap file, and merges them if they have the same pair of alleles (not necessarily in the same major/minor order) and if their mismatch rate is no greater than the threshold specified by **-maxMisMat**. If **-callHets** is on, then genotypic disagreements will be called heterozygotes; otherwise they will be set to missing (callHets is off by default).

Input:

- Input HapMap file(s). Use a plus sign (+) as a wild card character to specify multiple chromosome numbers (each chromosome in a separate file).

Output:

- HapMap files (one per chromosome) in which duplicate SNPs have been merged

Arguments:

MergeDuplicateSNPsPlugin	
-hmp	Input HapMap file(s). Use a plus sign (+) as a wild card character to specify multiple chromosome numbers (each chromosome in a separate file).
-o	Output HapMap file(s). Use a plus sign (+) as a wild card character to specify multiple chromosome numbers (each chromosome in a separate file).
-misMat	Threshold genotypic mismatch rate above which the duplicate SNPs won't be merged. Default: 0.05
-callHets	When two genotypes at a replicate SNP disagree for a taxon, call it a heterozygote. Defaults to false (=set to missing).
-kpUnmergDups	When a pair of duplicate SNPs are not merged (because they have different alleles or too many mismatches), keep them. Defaults to false (=delete them).
-s	Start chromosome. Default: 1
-e	End chromosome. Default: 10

Example command:

```
/programs/tassel/run_pipeline.pl -fork1 -MergeDuplicateSNPsPlugin
-hmp hapmap/unfilt/myStudy.c+.hmp.txt
-o hapmap/mergedSNPs/myStudy.mergedSNPs.c+.hmp.txt –misMat 0.1 –callHets
-s 1 -e 12 -endPlugin -runfork1
```

Gory Details:

This step should be run directly after TagsToSNPByAlignmentMTPlugin, using the HapMap file(s) from that step as input.

If the germplasm is not fully inbred, and still contains residual heterozygosity (like the maize NAM or IBM populations do) then **-callHets** should be on and **-maxMisMat** should be set fairly high (0.1 to 0.2, or even higher, depending on the amount of heterozygosity). Because the sequencing coverage is usually less than 1x, most of the time only one allele at a heterozygous SNP will be detected (particularly for *Ape*KI). Hence, duplicate SNPs genotypes from a true heterozygote may disagree simply because different alleles were sampled by the duplicate assays. Hence, these disagreements are not necessarily errors, and should not necessarily be used to prevent duplicate SNPs from being merged (unless your germplasm *is* highly inbred, with very little residual heterozygosity).

Indels (gaps) are ignored by this plugin: it makes no attempt to merge apparent duplicate gaps with the same chromosomal position.

<u>**GBSHapMapFiltersPlugin**</u>

Summary:

Reads HapMap format genotype files (one per chromosome) and filters out SNPs with low taxon coverage (missing data at most taxa), high heterozygosity, low (and/or high) minor allele frequency, and (optionally) that are not in LD with at least one neighboring SNP. Taxa with low SNP coverage (missing data at most SNPs) can also be removed. All cutoffs are adjustable except for the LD cutoff (Bonferroni corrected *p*-value <0.01).

Input:

- Key file
- Directory (folder) containing qseq files

Output:
- Directory (folder) containing a corresponding tagCount file for every qseq file in the input directory

Arguments:

<u>GBSHapMapFiltersPlugin</u>	
-hmp	Input HapMap file. Use a plus sign (+) as a wild card character to specify multiple chromosome numbers (each chromosome in a separate file).
-o	Output HapMap file. Use a plus sign (+) as a wild card character to specify multiple chromosome numbers (each chromosome in a separate file).
-mnTCov	Minimum taxon coverage, i.e. the minimum SNP call rate for a taxon to be included in the output, where call rate is the proportion of the SNP genotypes for a taxon that are not “N” (where N = missing). Default: 0.1
-mnScov	Minimum site coverage, i.e. the minimum call rate for a SNP to be included in the output, where call rate is the proportion of the taxon genotypes for that SNP that are not “N” (where N = missing). Default: 0.1
-mnF	Minimum value of F (inbreeding coefficient). Not tested by default. **Do not invoke this option unless you are working with inbred lines or an inbreeding species**.
-mnMAF	Minimum minor allele frequency Default: 0.0 (no filtering).
-mxMAF	Maximum minor allele frequency. Default: 1.0 (no filtering).
-hLD	Specifies that samples should be filtered for high LD. Default: false (off).
-sC	Start chromosome. Default: 1
-eC	End chromosome. Default: 10

Example command:
```
/programs/tassel/run_pipeline.pl -fork1 -GBSHapMapFiltersPlugin
-hmp hapmap/mergedSNPs/myStudy.mergedSNPs.c+.hmp.txt
–o hapmap/filt/myStudy.mergedSNPs.filt.c+.hmp.txt -hLD -mnTCov 0.05
-mnSCov 0.05 -sC 1 -eC 12 -endPlugin -runfork1
```

Gory Details:
If your study germplasm are from a single biparental cross, then the **-hLD** (high LD) filter (off by default) can be very useful to filter out bad SNPs with high genotyping error or incorrect physical genomic positions. To pass through the LD filter, a SNP must be in statistically significant LD (Bonferroni corrected *p*-value < 0.01) with at least one SNP that is a minimum of 128 bp away (*i.e.*, not from the same tag alignment or cut site) but within a window of 50 SNPs on either side. Currently, none of these LD parameters are adjustable.

<u>BiParentalErrorCorrectionPlugin</u>

Summary:
This documentation is yet to be written.

Input:
- This documentation is yet to be written

Output:
- This documentation is yet to be written

Arguments:

Example command:
```
/programs/tassel/run_pipeline.pl -fork1 -BiParentalErrorCorrectionPlugin
-endPlugin -runfork1
```

Gory Details:
This documentation is yet to be written.

<u>MergeIdenticalTaxaPlugin</u>

Summary:
This documentation is yet to be written.

Input:
- This documentation is yet to be written

Output:
- This documentation is yet to be written

Arguments:

Example command:
```
/programs/tassel/run_pipeline.pl -fork1 –MergeIdenticalTaxaPlugin
-endPlugin -runfork1
```

Gory Details:
This documentation is yet to be written.

<u>BinaryToTextPlugin</u>

Summary:
Reads a binary GBS file and outputs the equivalent text file.

Input:
- Binary File

Output:
- Text File

Arguments:

<u>BinaryToTextPlugin</u>	
-i <filename>	Input binary file name.
-o <filename>	Output text file name.
-t <type>	Type of input file (TagCounts, TBTBit, TBTByte, TOPM).

Example commands:
```
/programs/tassel/run_pipeline.pl -fork1 -BinaryToTextPlugin
–i tagCounts/rice.cnt -o tagCounts/rice_cnt.txt -t TagCounts
-endPlugin -runfork1
```

```
/programs/tassel/run_pipeline.pl -fork1 -BinaryToTextPlugin
-i tbt/rice.tbt.bin -o tbt/rice_tbt.txt -t TBTBit
-endPlugin -runfork1

/programs/tassel/run_pipeline.pl -fork1 -BinaryToTextPlugin
-i topm/rice.topm.bin -o topm/rice_topm.txt -t TOPM
-endPlugin -runfork1
```

TextToBinaryPlugin

Summary:
Reads a Text GBS File and outputs the equivalent binary file.

Input:
- Text File

Output:
- Binary File

Arguments:

TextToBinaryPlugin	
-i <filename>	Input text file name.
-o <filename>	Output binary file name.
-t <type>	Type of file (TagCounts, TBTBit, TBTByte, TOPM).

Example commands:
```
/programs/tassel/run_pipeline.pl -fork1 -TextToBinaryPlugin
–i tagCounts/rice.txt -o tagCounts/rice_cnt.cnt -t TagCounts
-endPlugin -runfork1

/programs/tassel/run_pipeline.pl -fork1 -TextToBinaryPlugin
-i tbt/rice.tbt.txt -o tbt/rice_tbt.bin -t TBTBit
-endPlugin -runfork1

/programs/tassel/run_pipeline.pl -fork1 -TextToBinaryPlugin
-i topm/rice.topm.txt -o topm/rice_topm.bin -t TOPM
-endPlugin -runfork1
```

Appendix 1: Key file example

The barcode key file is formatted as tab-delimited text. You can create it from Excel if you save it as tab-delimited text. In the example key below there are two lanes, each at 96 plex. The barcodes correspond to our 96-plex *Ape*KI layout. You can combine lanes from multiple flow cells in a single key file and GBS analysis if you wish. Note that there is a “Blank” in each plate, in different positions (H12 and H11). This facilitates diagnosis of accidental plate swaps. Only the first 7 columns are mandatory. You can add additional columns to the key file as you see fit – these will be ignored by the pipeline. **The sample names must not contain spaces or colons (‘:’)**. However, it is OK to include dashes, parentheses, or underscores: - ( ) _.

Flowcell	Lane	Barcode	Sample	PlateName	Row	Column
ABC12AAXX	1	CTCC	MySample001	MyPlate1	A	1
ABC12AAXX	1	TGCA	MySample002	MyPlate1	A	2
ABC12AAXX	1	ACTA	MySample003	MyPlate1	A	3
ABC12AAXX	1	CAGA	MySample004	MyPlate1	A	4
ABC12AAXX	1	AACT	MySample005	MyPlate1	A	5
ABC12AAXX	1	GCGT	MySample006	MyPlate1	A	6
ABC12AAXX	1	TGCGA	MySample007	MyPlate1	A	7
ABC12AAXX	1	CGAT	MySample008	MyPlate1	A	8
ABC12AAXX	1	CGCTT	MySample009	MyPlate1	A	9
ABC12AAXX	1	TCACC	MySample010	MyPlate1	A	10
ABC12AAXX	1	CTAGC	MySample011	MyPlate1	A	11
ABC12AAXX	1	ACAAA	MySample012	MyPlate1	A	12
ABC12AAXX	1	TTCTC	MySample013	MyPlate1	B	1
ABC12AAXX	1	AGCCC	MySample014	MyPlate1	B	2
ABC12AAXX	1	GTATT	MySample015	MyPlate1	B	3
ABC12AAXX	1	CTGTA	MySample016	MyPlate1	B	4
ABC12AAXX	1	ACCGT	MySample017	MyPlate1	B	5
ABC12AAXX	1	GTAA	MySample018	MyPlate1	B	6
ABC12AAXX	1	GGTTGT	MySample019	MyPlate1	B	7
ABC12AAXX	1	CCAGCT	MySample020	MyPlate1	B	8
ABC12AAXX	1	TTCAGA	MySample021	MyPlate1	B	9
ABC12AAXX	1	TAGGAA	MySample022	MyPlate1	B	10
ABC12AAXX	1	GCTCTA	MySample023	MyPlate1	B	11
ABC12AAXX	1	CCACAA	MySample024	MyPlate1	B	12
ABC12AAXX	1	GCTTA	MySample025	MyPlate1	C	1
ABC12AAXX	1	CTTCCA	MySample026	MyPlate1	C	2
ABC12AAXX	1	GAGATA	MySample027	MyPlate1	C	3
ABC12AAXX	1	ATGCCT	MySample028	MyPlate1	C	4
ABC12AAXX	1	TATTTTT	MySample029	MyPlate1	C	5
ABC12AAXX	1	CTTGCTT	MySample030	MyPlate1	C	6
ABC12AAXX	1	ATGAAAC	MySample031	MyPlate1	C	7
ABC12AAXX	1	AAAAGTT	MySample032	MyPlate1	C	8
ABC12AAXX	1	GAATTCA	MySample033	MyPlate1	C	9
ABC12AAXX	1	GAACTTC	MySample034	MyPlate1	C	10
ABC12AAXX	1	GGACCTA	MySample035	MyPlate1	C	11
ABC12AAXX	1	GTCGATT	MySample036	MyPlate1	C	12
ABC12AAXX	1	AACGCCT	MySample037	MyPlate1	D	1
ABC12AAXX	1	AATATGC	MySample038	MyPlate1	D	2
ABC12AAXX	1	ACGACTAC	MySample039	MyPlate1	D	3
ABC12AAXX	1	GGTGT	MySample040	MyPlate1	D	4
ABC12AAXX	1	TAGCATGC	MySample041	MyPlate1	D	5
ABC12AAXX	1	AGTGGA	MySample042	MyPlate1	D	6
ABC12AAXX	1	TAGGCCAT	MySample043	MyPlate1	D	7
ABC12AAXX	1	TGCAAGGA	MySample044	MyPlate1	D	8
ABC12AAXX	1	TGGTACGT	MySample045	MyPlate1	D	9
ABC12AAXX	1	TCTCAGTC	MySample046	MyPlate1	D	10
ABC12AAXX	1	CCGGATAT	MySample047	MyPlate1	D	11
ABC12AAXX	1	CGCCTTAT	MySample048	MyPlate1	D	12
ABC12AAXX	1	AGGC	MySample049	MyPlate1	E	1
ABC12AAXX	1	GATC	MySample050	MyPlate1	E	2
ABC12AAXX	1	TCAC	MySample051	MyPlate1	E	3
ABC12AAXX	1	AGGAT	MySample052	MyPlate1	E	4
ABC12AAXX	1	ATTGA	MySample053	MyPlate1	E	5
ABC12AAXX	1	CATCT	MySample054	MyPlate1	E	6

Flowcell	Lane	Barcode	Sample	PlateName	Row	Column
ABC12AAXX	1	CCTAC	MySample055	MyPlate1	E	7
ABC12AAXX	1	GAGGA	MySample056	MyPlate1	E	8
ABC12AAXX	1	GGAAC	MySample057	MyPlate1	E	9
ABC12AAXX	1	GTCAA	MySample058	MyPlate1	E	10
ABC12AAXX	1	TAATA	MySample059	MyPlate1	E	11
ABC12AAXX	1	TACAT	MySample060	MyPlate1	E	12
ABC12AAXX	1	TCGTT	MySample061	MyPlate1	F	1
ABC12AAXX	1	ACCTAA	MySample062	MyPlate1	F	2
ABC12AAXX	1	ATATGT	MySample063	MyPlate1	F	3
ABC12AAXX	1	ATCGTA	MySample064	MyPlate1	F	4
ABC12AAXX	1	CATCGT	MySample065	MyPlate1	F	5
ABC12AAXX	1	CGCGGT	MySample066	MyPlate1	F	6
ABC12AAXX	1	CTATTA	MySample067	MyPlate1	F	7
ABC12AAXX	1	GCCAGT	MySample068	MyPlate1	F	8
ABC12AAXX	1	GGAAGA	MySample069	MyPlate1	F	9
ABC12AAXX	1	GTACTT	MySample070	MyPlate1	F	10
ABC12AAXX	1	GTTGAA	MySample071	MyPlate1	F	11
ABC12AAXX	1	TAACGA	MySample072	MyPlate1	F	12
ABC12AAXX	1	TGGCTA	MySample073	MyPlate1	G	1
ABC12AAXX	1	ACGTGTT	MySample074	MyPlate1	G	2
ABC12AAXX	1	ATTAATT	MySample075	MyPlate1	G	3
ABC12AAXX	1	ATTGGAT	MySample076	MyPlate1	G	4
ABC12AAXX	1	CATAAGT	MySample077	MyPlate1	G	5
ABC12AAXX	1	CGCTGAT	MySample078	MyPlate1	G	6
ABC12AAXX	1	CGGTAGA	MySample079	MyPlate1	G	7
ABC12AAXX	1	CTACGGA	MySample080	MyPlate1	G	8
ABC12AAXX	1	GCGGAAT	MySample081	MyPlate1	G	9
ABC12AAXX	1	TAGCGGA	MySample082	MyPlate1	G	10
ABC12AAXX	1	TCGAAGA	MySample083	MyPlate1	G	11
ABC12AAXX	1	TCTGTGA	MySample084	MyPlate1	G	12
ABC12AAXX	1	TGCTGGA	MySample085	MyPlate1	H	1
ABC12AAXX	1	AACCGAGA	MySample086	MyPlate1	H	2
ABC12AAXX	1	ACAGGGAA	MySample087	MyPlate1	H	3
ABC12AAXX	1	ACGTGGTA	MySample088	MyPlate1	H	4
ABC12AAXX	1	CCATGGGT	MySample089	MyPlate1	H	5
ABC12AAXX	1	CGCGGAGA	MySample090	MyPlate1	H	6
ABC12AAXX	1	CGTGTGGT	MySample091	MyPlate1	H	7
ABC12AAXX	1	GCTGTGGA	MySample092	MyPlate1	H	8
ABC12AAXX	1	GGATTGGT	MySample093	MyPlate1	H	9
ABC12AAXX	1	GTGAGGGT	MySample094	MyPlate1	H	10
ABC12AAXX	1	TATCGGGA	MySample095	MyPlate1	H	11
ABC12AAXX	1	TTCCTGGA	Blank	MyPlate1	H	12
ABC12AAXX	2	CTCC	MySample096	MyPlate2	A	1
ABC12AAXX	2	TGCA	MySample097	MyPlate2	A	2
ABC12AAXX	2	ACTA	MySample098	MyPlate2	A	3
ABC12AAXX	2	CAGA	MySample099	MyPlate2	A	4
ABC12AAXX	2	AACT	MySample100	MyPlate2	A	5
ABC12AAXX	2	GCGT	MySample101	MyPlate2	A	6
ABC12AAXX	2	TGCGA	MySample102	MyPlate2	A	7
ABC12AAXX	2	CGAT	MySample103	MyPlate2	A	8
ABC12AAXX	2	CGCTT	MySample104	MyPlate2	A	9
ABC12AAXX	2	TCACC	MySample105	MyPlate2	A	10
ABC12AAXX	2	CTAGC	MySample106	MyPlate2	A	11
ABC12AAXX	2	ACAAA	MySample107	MyPlate2	A	12
ABC12AAXX	2	TTCTC	MySample108	MyPlate2	B	1

Flowcell	Lane	Barcode	Sample	PlateName	Row	Column
ABC12AAXX	2	AGCCC	MySample109	MyPlate2	B	2
ABC12AAXX	2	GTATT	MySample110	MyPlate2	B	3
ABC12AAXX	2	CTGTA	MySample111	MyPlate2	B	4
ABC12AAXX	2	ACCGT	MySample112	MyPlate2	B	5
ABC12AAXX	2	GTAA	MySample113	MyPlate2	B	6
ABC12AAXX	2	GGTTGT	MySample114	MyPlate2	B	7
ABC12AAXX	2	CCAGCT	MySample115	MyPlate2	B	8
ABC12AAXX	2	TTCAGA	MySample116	MyPlate2	B	9
ABC12AAXX	2	TAGGAA	MySample117	MyPlate2	B	10
ABC12AAXX	2	GCTCTA	MySample118	MyPlate2	B	11
ABC12AAXX	2	CCACAA	MySample119	MyPlate2	B	12
ABC12AAXX	2	GCTTA	MySample120	MyPlate2	C	1
ABC12AAXX	2	CTTCCA	MySample121	MyPlate2	C	2
ABC12AAXX	2	GAGATA	MySample122	MyPlate2	C	3
ABC12AAXX	2	ATGCCT	MySample123	MyPlate2	C	4
ABC12AAXX	2	TATTTTT	MySample124	MyPlate2	C	5
ABC12AAXX	2	CTTGCTT	MySample125	MyPlate2	C	6
ABC12AAXX	2	ATGAAAC	MySample126	MyPlate2	C	7
ABC12AAXX	2	AAAAGTT	MySample127	MyPlate2	C	8
ABC12AAXX	2	GAATTCA	MySample128	MyPlate2	C	9
ABC12AAXX	2	GAACTTC	MySample129	MyPlate2	C	10
ABC12AAXX	2	GGACCTA	MySample130	MyPlate2	C	11
ABC12AAXX	2	GTCGATT	MySample131	MyPlate2	C	12
ABC12AAXX	2	AACGCCT	MySample132	MyPlate2	D	1
ABC12AAXX	2	AATATGC	MySample133	MyPlate2	D	2
ABC12AAXX	2	ACGACTAC	MySample134	MyPlate2	D	3
ABC12AAXX	2	GGTGT	MySample135	MyPlate2	D	4
ABC12AAXX	2	TAGCATGC	MySample136	MyPlate2	D	5
ABC12AAXX	2	AGTGGA	MySample137	MyPlate2	D	6
ABC12AAXX	2	TAGGCCAT	MySample138	MyPlate2	D	7
ABC12AAXX	2	TGCAAGGA	MySample139	MyPlate2	D	8
ABC12AAXX	2	TGGTACGT	MySample140	MyPlate2	D	9
ABC12AAXX	2	TCTCAGTC	MySample141	MyPlate2	D	10
ABC12AAXX	2	CCGGATAT	MySample142	MyPlate2	D	11
ABC12AAXX	2	CGCCTTAT	MySample143	MyPlate2	D	12
ABC12AAXX	2	AGGC	MySample144	MyPlate2	E	1
ABC12AAXX	2	GATC	MySample145	MyPlate2	E	2
ABC12AAXX	2	TCAC	MySample146	MyPlate2	E	3
ABC12AAXX	2	AGGAT	MySample147	MyPlate2	E	4
ABC12AAXX	2	ATTGA	MySample148	MyPlate2	E	5
ABC12AAXX	2	CATCT	MySample149	MyPlate2	E	6
ABC12AAXX	2	CCTAC	MySample150	MyPlate2	E	7
ABC12AAXX	2	GAGGA	MySample151	MyPlate2	E	8
ABC12AAXX	2	GGAAC	MySample152	MyPlate2	E	9
ABC12AAXX	2	GTCAA	MySample153	MyPlate2	E	10
ABC12AAXX	2	TAATA	MySample154	MyPlate2	E	11
ABC12AAXX	2	TACAT	MySample155	MyPlate2	E	12
ABC12AAXX	2	TCGTT	MySample156	MyPlate2	F	1
ABC12AAXX	2	ACCTAA	MySample157	MyPlate2	F	2
ABC12AAXX	2	ATATGT	MySample158	MyPlate2	F	3
ABC12AAXX	2	ATCGTA	MySample159	MyPlate2	F	4
ABC12AAXX	2	CATCGT	MySample160	MyPlate2	F	5
ABC12AAXX	2	CGCGGT	MySample161	MyPlate2	F	6
ABC12AAXX	2	CTATTA	MySample162	MyPlate2	F	7
ABC12AAXX	2	GCCAGT	MySample163	MyPlate2	F	8

Flowcell	Lane	Barcode	Sample	PlateName	Row	Column
ABC12AAXX	2	GGAAGA	MySample164	MyPlate2	F	9
ABC12AAXX	2	GTACTT	MySample165	MyPlate2	F	10
ABC12AAXX	2	GTTGAA	MySample166	MyPlate2	F	11
ABC12AAXX	2	TAACGA	MySample167	MyPlate2	F	12
ABC12AAXX	2	TGGCTA	MySample168	MyPlate2	G	1
ABC12AAXX	2	ACGTGTT	MySample169	MyPlate2	G	2
ABC12AAXX	2	ATTAATT	MySample170	MyPlate2	G	3
ABC12AAXX	2	ATTGGAT	MySample171	MyPlate2	G	4
ABC12AAXX	2	CATAAGT	MySample172	MyPlate2	G	5
ABC12AAXX	2	CGCTGAT	MySample173	MyPlate2	G	6
ABC12AAXX	2	CGGTAGA	MySample174	MyPlate2	G	7
ABC12AAXX	2	CTACGGA	MySample175	MyPlate2	G	8
ABC12AAXX	2	GCGGAAT	MySample176	MyPlate2	G	9
ABC12AAXX	2	TAGCGGA	MySample177	MyPlate2	G	10
ABC12AAXX	2	TCGAAGA	MySample178	MyPlate2	G	11
ABC12AAXX	2	TCTGTGA	MySample179	MyPlate2	G	12
ABC12AAXX	2	TGCTGGA	MySample180	MyPlate2	H	1
ABC12AAXX	2	AACCGAGA	MySample181	MyPlate2	H	2
ABC12AAXX	2	ACAGGGAA	MySample182	MyPlate2	H	3
ABC12AAXX	2	ACGTGGTA	MySample183	MyPlate2	H	4
ABC12AAXX	2	CCATGGGT	MySample184	MyPlate2	H	5
ABC12AAXX	2	CGCGGAGA	MySample185	MyPlate2	H	6
ABC12AAXX	2	CGTGTGGT	MySample186	MyPlate2	H	7
ABC12AAXX	2	GCTGTGGA	MySample187	MyPlate2	H	8
ABC12AAXX	2	GGATTGGT	MySample188	MyPlate2	H	9
ABC12AAXX	2	GTGAGGGT	MySample189	MyPlate2	H	10
ABC12AAXX	2	TATCGGGA	Blank	MyPlate2	H	11
ABC12AAXX	2	TTCCTGGA	MySample190	MyPlate2	H	12