***PoolParty* Pipeline Example**

by: Steven Micheletti
updated 6/2022 by Stuart C. Willis

Synopsis:
This exercise uses 150-bp PE FASTQ files from a fake diploid organism (found in the 'example' folder) to illustrate performance of the three modules of *PoolParty*. If interested in using *PoolParty*, this is a good starting point to ensure the pipeline will run on your system and to better understand output files.

We will use a hypothetical example to detect genomic differences between category A with phenotype "X" and category B with phenotype "Z." Specifically, we will be aligning 5 libraries (10 *fastq* files) to a small (9.3 mb) reference genome (*PP_genome.fa*)

PP_genome.fa	-Small genome with 3 anchored chromosomes and 5 scaffolds
P101_103_R1_PopA.fq.gz	-Individual 101, read 1, from pop "A"
P101_103_R2_PopA.fq.gz	-Individual 101, read 2, from pop "A"
P102_97_R1_PopA.fq.gz	-Individual 102, read 1, from pop "A"
P102_97_R2_PopA.fq.gz	-Individual 102, read 2, from pop "A"
P201_104_R1_PopB.fq.gz	-Individual 201, read 1, from pop "B"
P201_104_R2_PopB.fq.gz	-Individual 201, read 2, from pop "B"
P202_103_R1_PopB.fq.gz	-Individual 202, read 1, from pop "B"
P202_103_R2_PopB.fq.gz	-Individual 202, read 2, from pop "B"
P203_97_R1_PopB.fq.gz	-Individual 203, read 1, from pop "B"
P203_97_R2_PopB.fq.gz	-Individual 203, read 2, from pop "B"

PoolParty will accept compressed (gzipped) or uncompressed read files. Note that the content of the *fastq* files have been greatly reduced for example purposes. Therefore, we will be interpreting regions with very low coverage.

0) Dependency installation, file encoding, and understanding your $PATH

PoolParty requires a number of **dependencies** that should be specified in the *config* files or available in the $PATH (see more below). We highly recommend installing these dependencies using *conda*. See installation notes distributed with the *PoolParty* scripts (https://github.com/stuartwillis/poolparty). The *PoolParty* scripts and examples can be downloaded from GitHub as follows, and the README contains installation instructions (below, "> " reflects a SHELL command line; don't type it, only the rest):

```
> mkdir ~/bin
> cd ~/bin
> wget https://github.com/stuartwillis/poolparty/archive/refs/heads/main.zip
> unzip main.zip
> rm main.zip
> mv poolparty-main poolparty
```

ALL text files must have UNIX line encodings (as opposed to Mac or DOS/Windows), and *PoolParty* will throw errors if other line endings are encountered. It's a good idea to make sure any text file you intend to use (e.g. config files, samplelist files) has the proper line endings. This can be specified before/when saving in most text editors, or our *conda* installation instructions include utilities (*dos2unix*, *mac2unix*) that allow conversion on the command line, e.g.

```
> dos2unix samplelist.txt
```

What is this "$PATH" thing, anyway?

$PATH is the variable that contains directories in which the system is supposed to look for programs when called. It usually contains several directories, and the system searches for something in each directory in order, and only looks in subsequent directories if it didn't yet find it. There are standard directories that each user inherits at startup on most systems, such as /usr/local/bin, but it is also customizable by adding directories using a $PATH definition line to a hidden file in your home directory called .bash_profile (the preceding "." is what makes it hidden; list hidden files with *ls -a*). You can view your current path with

```
> echo $PATH
```

What it means for a directory to be in $PATH is that the user does not need to provide the directory structure where a program (executable) is found to call it; just type the program's name (e.g. 'samtools'). There may be several versions or copies of a program in one's $PATH, and only the first encountered in the order of directories listed in $PATH will be used. To see which copy of a program will be called, try using the following (e.g. to find 'samtools'):

```
> which samtools
```

There are several ways to make sure a program (or script) is in $PATH. The first is to add the directory containing the program to $PATH, either temporarily (using the EXPORT command) or permanently (or until the file is edited again) using .bash_profile. Another is to add a [symbolic] link to the program in its current directory (one not in $PATH) to a directory that is in $PATH, e.g. /usr/local/bin. Both have their advantages. We use the latter approach below (though the scripts can be used without adding to $PATH at all, as demonstrated). The only constraint to this latter technique is that only a single link of the same name can be added in any directory, just like any file.

FYI, one of the things that happens when you activate a *conda* environment is that it adds that environment's directories to your path, at the beginning, meaning that the conda versions will be called first. If you had a *PoolParty conda* environment activated that included samtools when you issued the above command, you should have seen something like "~/miniconda3/envs/poolparty_env/bin/samtools". As described below, a number of dependency programs for *PoolParty* will need to be in the path, which you can confirm with the *which* command, or provide the full directory location in the *config* files.

1) Genome Preparation

PoolParty requires a reference genome indexed for its dependencies. Whenever acquiring a new reference genome *fasta*, it is essential to first prepare the genome.

PPprep_genome_cl.sh contains the commands needed to prepare the fasta genome, *PP_genome.fa*, if the *bwa* and *samtools* dependencies are in the $PATH. To do this, make the directory containing the genome fasta the active directory (*cd* to that directory), and then the script takes the name of the genome fasta and the path to *picard.jar* on the command line, and automates indexing:

```
> bash ~/bin/poolparty/PPprep_genome_cl.sh PP_genome.fa  \
        ~/miniconda3/envs/poolparty_env/share/picard-2.20.2-0/picard.jar
```

Alternatively, you may run the commands independently. If you choose to do this, first index the genome with *bwa*. This later allows *bwa mem* to align reads to the genome:

```
> bwa index -a bwtsw PP_genome.fa
```

Next, we index the genome with *samtools*. This produces a *.fai* file with useful information about each chromosome/scaffold's size:

```
> samtools faidx PP_genome.fa
```

Finally, we create a dictionary with *Picard Tools*. This allows *Picard* to properly sort aligned reads (confirm the location of the picard.jar file and modify the following command as necessary):

```
> java -jar ~/miniconda3/envs/poolparty_env/share/picard-2.20.2-0/picard.jar \
        CreateSequenceDictionary \
        REFERENCE=PP_genome.fa  OUTPUT=PP_genome.fa.dict
```

After running the script or these separate commands a number of files with additional extensions will be produced: *.sa, .fai., .dict, .pac,.bwt, .ann, and .amb*. It is important that the genome name serves as a prefix to these the files (*e.g., PP_genome.fa.dict*).

2) PPalign

It is assumed that at this point that all dependencies have been installed. *PPalign* will report an error if it is unable to load any dependency. Some R packages are also required and an attempt will be made to install these automatically if not detected.

a) Creating the samplelist

In this example, a samplelist.txt has already been created. This file can be named "samplelist.txt", in which case PPalign will find it automatically, or this file can be specified in the *config* file under any name as SAMPLELIST (see Example2). This is useful if different subsets of data are to be analyzed from the same INDIR directory, or in different sample groups. It is assumed that both the read files and sample list file will be in the directory specified as INDIR in the *config* file. The sample list contains one *fastq* file per line with its corresponding sample group (*tab-delimited, with unix line endings, including a carriage return on the last line*). In this example, we have 5 libraries (paired read files) that belong two different sample groups (also called "population" by *PPalign*). A sample group can be known population assignment or a known trait. For this example, let's say that sample group 1 has phenotype "X" and sample group 2 has phenotype "Z".

Take a look at *samplelist.txt*:

```
> cat samplelist.txt

P101_103_R1_PopA.fq.gz          1
P101_103_R2_PopA.fq.gz          1
P102_97_R1_PopA.fq.gz           1
P102_97_R2_PopA.fq.gz           1
P201_104_R1_PopB.fq.gz          2
P201_104_R2_PopB.fq.gz          2
P202_103_R1_PopB.fq.gz          2
P202_103_R2_PopB.fq.gz          2
P203_97_R1_PopB.fq.gz           2
P203_97_R2_PopB.fq.gz           2
```

The naming convention of the *fastq* files themselves is somewhat flexible but has some requirements. The text before the first underscore must be a unique identifier that is shared between paired read files (library). In this example, we have two *fastqs* for each unique library: 101, 102, 201, 202, and 203. In this example, after the unique ID, optional text that denotes lane information is present. Within the remaining text, the file must be designated as read 1 (R1 or similar) or read 2 (R2 or similar) to differentiate the paired-end reads. Optional text following the first underscore may include a population field or other designators, before the file extension. The number of underscore delimited fields must be the same in each paired-read file name, and this text should be the same, except the read 1/2 designation. The file extension should follow common *fastq* conventions such as *.fastq, fq., fastq.gz, etc.*

Next to each filename, *PPalign* will recognize shared sample group assignment (1 or 2 in this case, which corresponds to phenotype "X" and "Z") and merge these alignments at a later step. ~~Category assignment can be an integer and range from 1 to the number of sample groups.~~ **

More examples of file names that will work:

```
Pop1_FR.fq      1
Pop1_RR.fq      1
452_202i_1.fastq        2
452_202i_2.fastq        2
```

Examples of file names that WILL NOT work below. In this second case, *PPalign* sees only one

individual (603). The script will exit with an error:

```
603_ind1_R1.fq      1
603_ind1_R2.fq      1
603_ind2_R1.fq      2
603_ind2_R2.fq      2
```

In this third case, *PPalign* will not understand that these are matching reads because read 2 has excess information that is not seen in read 1. The script should exit with an error:

```
POP1_R1.fastq       1
POP1_102_R2.fastq   1
```

**SCW Update note: I have declined to change sample group designations in this example file to ensure compatibility with previous tutorial results. However, it is important to note that using solely numeric designations (e.g. 1-N sample groups) causes confusion, both in how samples will be sorted in the *PPalign* output (alphanumerically, and Linux sorts 1, 2, 3,…10 differently than 01, 02, 03,…10, i.e. as 1, 10, 2, 3 versus 1, 2, 3, 10) as well as what should be specified in the *PPanalyze config* file (column ranks of desired samples, not sample name/number designations) and what should be specified for the CMH test (re-ordered column ranks for category replicates in the subset *sync* file created by *PPanalyze*). Thus, we recommend that sample groups should be specified as alpha (A, B, C, etc.), alphanumeric codes (e.g. 02_LEW, 03_SKA, 04_WIL, etc.), or at the very least numeric codes with leading zeros for every digit (e.g. 001, 002,…010,…121), though the latter is the least preferred for the aforementioned reasons (see more explanation later). Otherwise, sample group designations need only be unique and sortable under Linux conventions.

If you are uncertain how your sample group designations will be interpreted and sorted, use the following command on your sample list file to see what will be interpreted by *PPalign*, e.g.:

```
> cut -f2 samplelist.txt | sort | uniq
```

b) Editing the config file.

The configuration file contains directory locations, parameters, and dependency locations. You will need to edit *pp_align.config* (at least) to tailor directories to your system. Note, that there cannot be any spaces between "=" and the definition of the variable. The script will exit with an error if so.

Open *pp_align.config*

```
> cat pp_align.config
```

```
#Input/Output#
INDIR=/usr/local/bin/poolparty/example
        Change this to the directory where the example fastq files and samplelist.txt are.
SAMPLELIST=samplelist-config.txt
OUTDIR=/userdir/phenoAB
        Change this to a directory where you have permission to write files to.
OUTPOP=phenoAB
        You may leave this as is or change it to a more familiar name. This will be the prefix
        to most files that are written.
GENOME=/usr/local/bin/poolparty/example/PP_genome.fa
        Specify the genome name with its path.
SCAHEAD=Scaff
        If you open PP_genome.fa.fai you will see the three chromosomes and their lengths in
        basepairs. In this reference genome, there are 5 scaffolds are noted by Scaff. By
        indicating the scaffold heading, we are telling PPalign how to differentiate chromosomes
```

from scaffolds. This is optional. Leave this as is for this example.

#Run Parameters#*

THREADZ=4

Specific the maximum number of threads to use. Some of the commands and dependencies of *PoolParty* utilize multi-threading, in particular the mapping utility (bwa) and SNP calling with bcftools. Practically speaking, using more processors than the number of chromosomes (for those genomes with designated chromosomes) will not provide any meaningful increase in speed, but could cause crashes because of resource limitation conflicts. Four should be more than enough for this example.

KMEM=Xmx4g

Memory parameter for java-based programs. Memory issues with java can be common and this parameter may need to be tailored to your specific system. Xmx means the maximum size in the memory allocation pool, with 4g being 4gb. If unsure about optimal memory parameters leave as is.

BQUAL=20

Base quality refers to PHRED score given to each base in a read. Base quality tends to deteriorate towards the ends of reads and low-quality bases should be trimmed to reduce inaccurate base calls. *BBduk* from *BBmap* uses this quality to remove a read if its average base quality falls below this value, or trim a region of reads if their average quality is below this value.

MINLENGTH=25

After bases are trimmed by quality, the read may become so short that it cannot properly align to the reference genome. MINLENGTH is the minimum length a read must be after trimming to retain it. Base-ically, MINLENGTH=25 means discard any reads that fall below 25 bp after quality trimming.

MAPQ=5

Mapping quality is defined differently by various aligners. In general, it is a score denoting how well reads have aligned to the reference genome. Reads are penalized by factors such as mismatches. A MAPQ score of 0 means that reads are aligning to different portions of the genome with the same probability. With lower MAPQ, more reads will be retained but have lower confidence in alignment calls; higher MAPQ will retain fewer reads, but with higher confidence. For this example, we will use a low MAPQ of 5.

SNPQ=20

SNP quality (QUAL) will depend on the package calling SNPs. In this case, *bcftools* will provide a QUAL which is confidence in that the genomic position is indeed a SNP. For instance, a putative heterozygous position may be a variant, or perhaps all individuals are fixed for the heterozygous genotype. *bcftools* will take into account read depth at the reference and alternate alleles for all libraries and provide QUAL score. The main purpose of this filter in *PPalign* is to reduce file sizes and thus memory usage. A SNPQ of 20 removes highly unlikely SNPs; however, this value may want to be raised or lowered based on the purpose of the experiment. For this example, we will use SNPQ=20.

MINDP=10

The minimum depth of coverage summed across all libraries required to retain a SNP. Depth of coverage filters are applied during the analysis step as well; this serves as an initial screening step to reduce the number of SNPs and thus file sizes. For this example, we will use a MINDP of 10, though this value should NOT be treated as a default.

MAF=0.05

Global minor allele frequency. Removes SNP calls where the minor allele is below MAF. Will greatly reduce the number of uninformative SNPS and thus file sizes.

INWIN=15

Alignment error can occur around indels leading to false-positive SNPs being called in these regions. INWIN is the number of basepairs around an indel to discard when calling SNPs. This is a quick alternative to local realignment. For this example, let's keep INWIN at 15, which is a large region for indel-region SNP removal.

#Run-types#
SPLITDISC=off

Samblaster can identify split-end and discordant aligned reads and output them as separate *sam* files. These read types are useful when investigating structure variants through atypical alignment of reads. SPLITDISC=off will skip this production which can save computation time and space. Leave off for example.

INDCONT=on

If individuals (or multiple libraries) per sample grouping were included, INCONT=on will run scripts that check for variance in allele contribution of SNPs and produce individual stats. Additionally, a normalization process will occur whereby each individual's allelic contribution to a SNP is normalized (see explanation below). Note that this process is memory intensive, and memory usage can be managed with the DIVIDE parameter (see explanation below and demonstration in Example2).

QUALREPORT=on

fastqc provides helpful quality plots and stats on trimmed reads. Sometimes quality reports are not needed and QUALREPORT=off will prevent *fastqc* from running. Leave on for example

#Dependency Locations#
BCFTOOLS=bcftools #modify if not in the $PATH
FASTQC=fastqc #modify if not in the $PATH
BWA=bwa #modify if not in the $PATH
SAMBLASTER=samblaster #modify if not in the $PATH
SAMTOOLS=samtools #modify if not in the $PATH
PICARDTOOLS=/usr/local/bin/picard.jar #modify as needed
BBMAPDIR=/usr/local/bin/bbmap/ #modify as needed
POOL2=/usr/local/bin/popoolation2_1201/ #modify as needed

PPalign will ensure that all dependencies are installed and throw an error if not. Ensure that the correct command and/or directory is set for each dependency. Note that R, perl, and java languages are required as well.

c) Running PPalign

At this point we have a samplelist (containing sample group designations representing the study question) and a config file ready to go. We are now ready to run *PPalign*.

If you have administrative privileges, it is convenient to run *PPalign* directly from the command line (without specifying the directory). To do this, create a symbolic link to a location in the $PATH (e.g. /usr/local/bin, if your user has write privileges there) to where *PPalign.sh* was downloaded.

```
> ln -s ~/bin/poolparty/PPalign.sh /usr/local/bin/PPalign
```

To run *PPalign*, we simply pass the config file as an argument. Make sure you are in the directory where the config file is located and run *PPalign:*

```
> PPalign pp_align.config |& tee example_log.out
```

Alternatively, you can run all the PoolParty scripts without symbolic links as below:

```
> bash ~/bin/poolparty/PPalign.sh pp_align.config |& tee example_log.out
```

In each case this will write all alerts to *log.align_example.txt* in the current directory (while also displaying to screen), which contains a lot of useful information. You may also run the command without "|& tee example_log.out" but alerts will not be saved to a file. Note to either use the Linux *screens* utility (preferred) or *ctrl-z*, *bg*, and *disown -h* to prevent the script from being terminated with *ssh* interruptions.

d) PPalign Results

If executed properly, thousands of lines of alerts will be written to the log file and output files will be written to the output directory. The whole alignment module should take <5 mins to finish in this example.

A copy of your config file with the run date is produced in the output folder. This allows you to continue to modify a single config file for different *PPalign* runs since the parameters for each run will be saved in the respective output folder.

Let's go through what is happening step by step:

i) The log file.

Using Unix commands, you can quickly query *PoolParty*-specific Alerts in the log file (*example_log.out*)

```
> grep "ALERT " example_log.out
```

The status of the run should be produced. Most analyses have a timestamp that gives an indication on how fast each portion of the module run.

If you suspect errors:

```
> grep "ERROR" example_log.out
```

If you are interested in the number of optical PCR duplicates produced:

```
> grep "duplicates" example_log.out
```

When you query the log file, the order of *fastqs* will be consistent. You can check on how the pipeline ordered names in the output folder:

```
> cat OUTDIR/phenoAB_names.txt

P101_103_R1_PopA
P102_97_R1_PopA
P201_104_R1_PopB
P202_103_R1_PopB
P203_97_R1_PopB
```

This is the order that the *fastqs* are processed and combined for relevant downstream analyses.

```
> cat OUTDIR/pops/pop_1.txt
```

Shows you which libraries (individuals) belong to population 1.

ii) BBduk (BBmap suite) quality trimming.

BBduk performs the first step of the pipeline by quality trimming, kmer trimming, and contaminant (adapters, dimer, *etc.*) removal.

```
> grep "Low quality" example_log.out
```

Describes the number low-quality reads and bases removed :

```
Low quality discards:    3344 reads (3.60%)    339351 bases (2.57%)
Low quality discards:    3006 reads (2.56%)    305700 bases (1.82%)
Low quality discards:    1660 reads (2.41%)    152345 bases (1.57%)
Low quality discards:    1564 reads (2.32%)    139202 bases (1.47%)
Low quality discards:    1052 reads (1.72%)    86567 bases (1.01%)
```

```
> grep "Contaminant" example_log.out
```

Indicates that no contaminants were detected. However, real sequencing data will likely have a proportion of contamination from adapters and primer dimers. *BBMap* contains a contaminant list in /resources/*adapters.fa* , which contains common dimer and adapter sequences. This list is fully customizable and you may add contaminant sequences based on the library preparation protocol you are using. For instance, if you are using a NebNext kit, you may want to look up NebNext primers and include them in this list.

We can see more information about trimmed reads in OUTPUT/trimmed/, which contains the trimmed *fastqs* and stats files for each sample. The trim stats have some basic information, like number of reads, but will also produce the putative source of contamination if any is detected.

iii) Fastqc quality checking

Fastqc in *PoolParty* checks for quality AFTER trimming, meaning that quality reports should show high- quality reads. OUTPUT/quality/ has html files with summary stats on each read.

If we open *P203_97_R1_PopB.trim_2_fastqc.html* we check summary of the quality:

Measure	Value
Filename	P203_97_R1_PopB.trim_2
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	27761
Sequences flagged as poor quality	0
Sequence length	25-151
%GC	42

In some cases, we may want to tune the BQUAL parameter to allow for more data, or filter out more low-quality reads.

iv) Read Alignment and bam filtering

Trimmed reads are aligned to the reference genome with *bwa mem* while duplicates are removed by *samblaster* and mapping quality is filtered by *samtools*.

Aligned bam files are then processed in three steps:

1) *aligned.bam* = trimmed reads aligned to the specified reference genome
   bams that have been aligned by *bwa mem*, filtered by MAPQ, and have PCR duplicates removed.

2) *sorted.bam* = aligned bam files that have been coordinate-sorted by *Picard bams* sorted by chromosome and position coordinates.
3) *filtered.bam* = sorted bam files that have been filtered by *samtools* to remove uninformative read information. Unpaired and unaligned reads are removed to reduce size of the bam files.

Note that after the filtered *bam* files are created, the aligned *bam* files are deleted to save space. Therefore, if we go into /OUTDIR/BAM, we should only see sorted and filtered *bam* files (the sorted bam files may be deleted at the user's discretion also).

After the *bam* files are aligned, *samtools* stat reports are also generated for each. These can be found in OUTDIR/reports. Let's look at one:

> cat *P203_97_R1_PopB_phenoAB_aln_report.txt*

```
43119 + 0 in total (QC-passed reads + QC-failed reads)
506 + 0 secondary
0 + 0 supplementary
0 + 0 duplicates
43119 + 0 mapped (100.00% : N/A)
42613 + 0 paired in sequencing
21285 + 0 read1
21328 + 0 read2
41263 + 0 properly paired (96.83% : N/A)
42172 + 0 with itself and mate mapped 441 + 0 singletons (1.03% : N/A)
105 + 0 with mate mapped to a different chr
105 + 0 with mate mapped to a different chr (mapQ>=5)
```

The total number of reads (43119) refers to the total number of reads *after* quality trimming. Secondary reads (506) are those that have a high chance of aligning to multiple positions in the ref genome. We should see 100% mapped (43119) every time since the report is performed on a bam that has been aligned to ref genome. However, we are generally interested in properly paired (41263) as these are the reads that have aligned with their mate in proper orientation and will be retained for SNP calling.

We also see *pop_1.bam* and *pop_2.bam*, which are merged *bam* files. All aligned reads with phenotype "X" are in *pop_1.bam*, and all aligned reads with phenotype "Z" are in *pop_2.bam*. In other words, *fastq* files with the same categorical designation (1 or 2 in this case) in *samplelist.txt* will be combined.

v) SNP calling

It can be a waste of computational power and space to process and store all aligned genomic positions (*i.e.,* monomorphic sites). Therefore, *PPalign* first uses *bcftools* to call all potential SNPs, then uses these SNPs to filter other file formats, essentially treating the SNP calls as whitelist.

bcftools uses the combined *bam* files (*pop_1.bam* and *pop_2.bam*) to predict all potential SNPs based on all individual libraries and creates a variant call format (VCF) file. Then, *bcftools* uses user-specified parameters to filter out unlikely and rare SNPs with the SNP quality (SNPQ), global minor allele frequency (MAF), and total depth of coverage (MINDP) filters specified in the config file. This not only reduces space, but will speed up downstream analyses.

The output reflects how many SNPS were called (values may vary slightly with dependency versions):

> grep "SNPs" example_log.out

```
ALERT: Samtools and bcftools are calling SNPs and creating mpileup at Thu Mar 8 15:57:17 MST
2018 ALERT: 217546 SNPs total SNPS called without filters
ALERT: 202716 SNPs removed due to QUAL < 20 and total DP < 10
ALERT: Additional 905 SNPs removed due to global MAF < 0.05
```

```
ALERT: 13925 total SNPs retained after SNP calling
ALERT: Of the remaining SNPs, there are 13731 SNPs and 194 INDels
```

An original VCF* file with 217,546 SNPs was produced; however, the majority of these SNPs were filtered out due to the various filters, leaving only 13,925 SNPs. We additionally see that of these SNPs, 194 are indels. *PPalign* will split the variants calls into two files in OURDIR/filters, one listing all indels and the other listing all SNPs, and transfer the most pertinent information from the filtered VCF file to $OUTFILE_variants.txt file. Except to preserve for calling the same SNPs on future aligned samples (using the USEVCF option; see Example2), the VCF files may be deleted to save disk space.

PPalign creates an mpileup file (*phenoAB.mpileup*) while simultaneously filtering it by the SNP positions above. This *mpileup* file is used for making a *sync* format that is required for downstream comparative analyses.

Another large *mpileup* is created for all aligned positions in each bam. This is generally a very large file, though quality information is stripped to reduce size (*phenoAB_stats.mpileup*). This file is coverage information for each category (pop_1 or pop_2) at each aligned genomic position and is required for *PPstats*. If stats aren't needed (or after they have been run) and disk storage is an issue, this file can be deleted.

phenoAB_variants.txt shows the reference chromosome and position for all the remaining SNPs with SNP quality and depth. Additionally, indels are noted with "IDV." This file allows quick and intuitive access to the SNPs you will analyzing.

*Note that if libraries correspond to individuals, it is possible to use the VCF file to extract individual genotypes. Using *bcftools* (or *vcftools*), you can filter individual sites that have sufficient depth of coverage (usually 10X minimum on average) to retain individual genotypes with high confidence. However, we recommend a more robust option is to utilize the bam files with the software *ANGSD*, which provide genotype probabilities and most likely genotypes. See information for utilizing *ANGSD* with *PPalign* output accompanying the *PoolParty* distribution.

vi) Sync creation

With the SNP-filtered *mpileup*, *PPalign* creates a *sync* format, which is a *Popoolation*-based format. This intuitive format has a column for each library with six digits separated by colons:

```
> head -n3 phenoAB.sync

Chr1   6722   G    1:0:0:4:0:0   1:0:0:0:0:0
Chr1   10101  C    0:1:1:0:0:0   0:3:3:0:0:0
Chr1   10104  T    0:1:1:0:0:0   0:3:3:0:0:0
```

The first column is the reference chromosome or scaffold name, the second is the position within the chromosome, and the third is the reference allele. Additional columns refer to allele counts for pop_1 and pop_2 whereby the order of the positions refer to bases: A T C G N del

For example, the first line of *phenoAB.sync* denotes that pop_1 has 1 copy of the A allele and 4 of the G allele, and pop_2 only has one copy of the A allele.

Popoolation2 uses the *mpileup* file to identify indels and remove SNPs that fall within our specified INWIN of 15 bp. This generally means a range of about 30 bp will be marked around indels and removed. This information is shown in phenoAB.gtffile

```
> head -n3 phenoAB.gtffile
```

```
Chr1  mpileup indelregion  5      34     .  .  .  gene_id "."; transcript_id ".";
Chr1  mpileup indelregion  10140  10181  .  .  .  gene_id "."; transcript_id ".";
Chr1  mpileup indelregion  39405  39434  .  .  .  gene_id "."; transcript_id ".";
```

This file specifies the genomic range it will mask based on indel detection and the INWIN. Not all ranges are the same size because often there are multiple indels within a region.

The *gtffile* file is then uses to remove the blacklisted indel regions and produce a final *.sync* file called *phenoAB.sync*. We can see how many SNPs were lost through the indel region masking process:

```
> grep "indel" example_log.out
```

```
ALERT: With an indel window of 15 bp you lost 1435 SNPs (11%)
```

11% may be a lot of data to throw away and we can certainly tune indel windows in future runs. The impact of indel regions on false-positive SNP-calling is expected to be small based on simulation and thus a small INWIN may be appropriate. If local realignment is a major concern, the commonly-used *GATK* package allows for local realignment around indel regions to retain all SNPs within these regions (not currently built into the pipeline).

vii) Frequency creation

After the final *sync* file (*phenoAB.sync*) is created, the R script, *r_frequency.R*, is initiated to estimate allele frequencies at each position. The format of this table is as follows:

```
> head -n3 phenoAB.fz
```

Chr	Pos	Ref	A1	1	2
Chr1	6722	G	A	0.2	1
Chr1	10101	C	T	0.5	0.5

The columns are chromosome (Chr), position (Pos), reference genome allele (Ref), reference empirical allele (A1), A1 frequency for pop_1 (1), and A1 frequency for pop_2 (2).

In row 1, pop_1 has an allele frequency of 20% A, 80% G, and pop_2 has an allele frequency of 100% A.

Note that A1 is not synonymous with an alternate allele, but rather the most common allele and the one to which the frequency values refer. When the Ref allele is the most common, Ref=A1, otherwise these may differ. If you want to know what the alternate allele is, this information is stored in the VCF file.

r_frequency.R uses frequencies to perform multiple filters, which will show up in the OUTDIR/filters directory. First, it performs a second minor allele frequency check at the specified MAF. This can be slightly different than the MAF filter performed on the VCF because libraries of the same category are merged in the *sync* file, yet split up in the VCF file. To see this,

```
> wc -l pops/phenoAB_MAF_fail.txt
```

indicates that just one additional SNP failed MAF when populations were combined

Next, the R script marks sites where >2 alleles were called. Though biologically possible between

populations of diploid organisms, in most cases this is a red flag for duplicated regions or paralogs. In the $OUTDIR/filters folder:

phenoAB_norm_poly_one.txt indicates positions where at least one population had >2 alleles
phenoAB_norm_poly_half.txt indicates positions where half of the populations had >2 alleles
phenoAB_norm_poly_all.txt indicates positions where all populations had >2 alleles

These lists can be incorporated into a blacklist for *PPanalyze* (see Example2).

Finally, the R script determines coverage for each population at each site in *phenoAB_coverage.txt*

viii) Individual analyses

With INDCONT=on, *PPalign* performs additional analyses and assumes that each *fastq* file that was provided belongs to a different individual. The main purpose is to understand each individual's contribution to each allele frequency estimate, provide additional filters, and normalize so that one or a few individuals do not bias allele frequency estimates (explained below).

It is worth noting that R scripts in these analyses can use a lot of memory, which is proportional to the number of individuals in each sample and the number of variants recorded (MxN). *PoolParty* now includes a feature to prevent crashes from high memory demands: the normalization of allele frequencies within a sample can be divided among subsets of individuals, and then summed. The DIVIDE parameter controls this subset group size. Normalization can be expected to take linearly longer with smaller values for DIVIDE, but a crashed run never finishes, so can be advantageous (see demonstration in Example2). INDCONT also creates large temporary *sync* files to count allelic contribution for each individual in the OUTDIR/inds folder; these may be deleted when the run completes.

*pop_*{POP}*_ind_stats.txt* indicates SNP contribution of each individual in each sample. Individuals are in the same order of how they are displayed in OUTDIR/pops. To see this:

```
> head -n3 inds/pop_2_ind_stats.txt
```

##Mean.c	Mean.non.z	Max.cov	std.non.ze	n.p	proportion.S
1.797	2.133	9	1.201	11	0.842
1.736	2.072	10	1.144	11	0.838
1.547	1.939	9	1.101	11	0.798

OUTDIR/pops/*pop_2.txt* indicates that in this example, the order of libraries in this sample is P201, P202, and P203.

The columns specify the mean coverage (Mean.cov), mean non-zero coverage (Mean.non.zero), maximum coverage (Max.cov), non-zero standard deviation of coverage (std.non.zero), number of SNPs represented (n.pos), and the proportion of SNPs represented of each individual (proportion.SNPs).

In this example, each library looks relatively uniform. On average they contribute similar mean coverage to each SNP they represent (~2 X), and cover a similar proportion of SNPs (~ 80%). If not uniform, there may be less confidence in allele frequency estimates and the normalized *sync* and *fz* files should be preferred.

*pop_*{POP}*_snp_stats.txt* indicates stats for each SNP in the respective sample/sample group.

```
> head -n4 inds/pop_1_snp_stats.txt
```

##CHR	pos	Mean.coverage	Sum.coverage	Number.individuals	Maximum.ind	Maximum.%.ind	Ind.%	Std
Chr1	19	1.5	3	2	2	0.667	1	0.707
Chr1	6722	2.5	5	2	4	0.8	1	2.121
Chr1	10101	1	2	1	2	1	0.5	NA

For each genomic position, we have the mean individual coverage (Mean.coverage), total coverage (Sum.coverage), number of individuals represented at that site (Number.individuals), maximum coverage contribution by a single individual (Maximum.ind), the proportion of the maximum individual contribution (Maximum.%.ind), percent of individuals represented at that site (Ind.%), and coverage standard deviation between represented individuals (Std).

*pop_*{POP}*_snp_stats.txt* files can be rather large and their main purpose is to be used to blacklist additional SNPs for downstream analyses. For instance, let's say we want to blacklist sites that are only represented by a single individual (e.g. for use in *PPanalyze*):

```
> awk '$5 < 2' inds/pop_1_snp_stats.txt | awk '{print $1,$2}' > filters/black_list.txt
```

This command filters the Number.individuals column ($5) by individuals less than 2, then pipes to only print the first 2 columns (Chr,Pos) and writes this out to a file called *black_list.txt.*

Or, perhaps we want to mark SNPs with a standard deviation that is greater than or equal to 2:

```
> awk '$9 >= 2 || $9 ==NA' inds/pop_1_snp_stats.txt | awk '{print $1,$2}' > filters/black_list.txt
```

Note that we specify that stdev is >= 2 OR (||) ==NA. NAs occur when no st.dev is present (*i.e.*, 1 individual). We don't have to write these SNPs to a file either; we can just check how many SNPs fail our criteria:

```
> awk '$9 >= 2 || $9 ==NA' inds/pop_1_snp_stats.txt | wc -l
```

Indicates 8939 SNPs would be removed if these criteria are used.

Use the *pop_*{POP}*_snp_stats.txt* files to build unique black lists based on what makes sense with your data and organism. We include a script, *PP_blacklist_minimum_individuals.sh*, that returns a blacklist of SNPs with less than the specified minimum number of individuals (or libraries) surveyed in each sample grouping (see demonstration in Example2).

Finally, INCONT=on performs a normalization step that produces a unique $OUTDIR_norm.*sync* file. This records normalization of the contribution of each individual, as follows:

Using a hypothetical example, say there are 4 individuals represented at a SNP:

IND1) 10 T 51 A
IND2) 5 T, 2 A
IND3) 30 T 1 A
IND4) 0 T 1 A

The method essentially corrects these individuals as such:

IND1) 1 T 1 A
IND2) 1 T 1 A
IND3) 2 T 0 A
IND4) 0 T 1 A

IND1 we are pretty sure is heterozygous, so it contributes one of each of its alleles.
IND2, though with low coverage, also appears to be heterozygous and now holds equal weight to IND1.
IND3 is technically sequenced at 2 alleles, but there is such a large imbalance in alleles we can't rule out contamination, and we assume homozygosity; thus, IND3 contributes a homozygous genotype of 2 T alleles.
IND4 has so few reads we cannot be certain of its genotype beyond including A; thus it contributes only a single A allele.

In this example the normalization essentially changes:

55 A (55%) and 45 T (45%) to 3 A (43%) and 4 T (57%)

If an individual only contributes a single allele (i.e., 1X of an allele), it will continue to only contribute 1 allele in this method. If an individual has >2 alleles at a site, its contribution will be discarded for that SNP. In the end, this is a pseudo-genotyping method that attempts to greatly reduce if not eliminate bias that occurs due to over-representation of a few individuals.

In the main OUTDIR folder, compare *phenoAB_norm.sync* (normalized sync file) to *phenoAB.sync* (standard synce file). There are also frequency tables produced for the normalized sync file (*i.e.*, OUTPOP_norm.fz) where you can determine how allele frequencies have changed.

3) PPstats

a) Prerequisites

PPstats is an optional module that produces some stats and plots to assess sequencing and alignment performance. It only requires *$OUTFILE_stats.mpileup* created by *PPalign* and the genome index* (*PP_genome.fa.fai*) to generate stats. Create another symbolic link if you want to run it directly from the commandline:

```
> $ ln -s ~/bin/poolparty/PPstats.sh /usr/local/bin/PPstats
```

*Note that naming convention of chromosomes and scaffolds currently has a large impact on the plotting of stat results. This is because the R plotting script attempts to split chromosomes and scaffolds and order them numerically. The naming of chromosomes in some NCBI assembly genomes, which contain periods, follows a format that is not yet supported. Ideally chromosomes and scaffolds would follow a format of Chr01, Chr02, Chr03, *etc.* whereby chromosomes are easily identifiable and sorted by their order in the genome. This aids plotting of results from *PPanalyze* also. We suggest that you consider modifying chromosome designations of empirical fasta files to facilitate this.

b) Editing the config file.

> cat *pp_stats.config*

#Specify Files#
FAI=/usr/local/bin/poolparty/example/PP_genome.fa.fai
The indexed genome file we previously generated, with path.
MPILEUP=/userdir/phenoAB/phenoAB_stats.mpileup
The large _stats.mpileup file we previously generated, with path.
OUTDIR=/userdir/phenoAB/stats
Choose a unique output directory for stats file
OUTFILE=PP_stats.txt
Choose a unique name for the stats output file

#Parameters#
SCAFP=Scaff
If there are both anchored chromosomes and unanchored scaffolds in the reference assembly, *PPstats* will attempt to differentiate the two. *PPstats* looks for scaffolds beginning with SCAFP and treats those as unanchored portions of the genome. In the example genome, scaffolds are denoted by "Scaff." If there are no scaffolds (or only scaffolds), leave this parameter empty.
THREADZ=5
PPstats will use multiple processors with low memory usage. Choose maximum number of parallel-processes to run at once.
MINCOV=2
Minimum depth of coverage to retain a genomic position. Primarily used to determine the breadth of the genome covered by high depth of coverage reads. Set very low for this example to MINCOV=2, which should not be treated as a default.
MAXCOV=250
Maximum depth of coverage to retain a genomic position. Set to 250 for this example, which should not be treated as a default.

c) Running PPstats

Switch directory containing the config file and run:

```
> PPstats pp_stats.config |& tee log.stats_example.txt
```

PPstats will provide rough status updates for the 8 blocks it runs through. The example should finish in less than a minute.

d) Output files

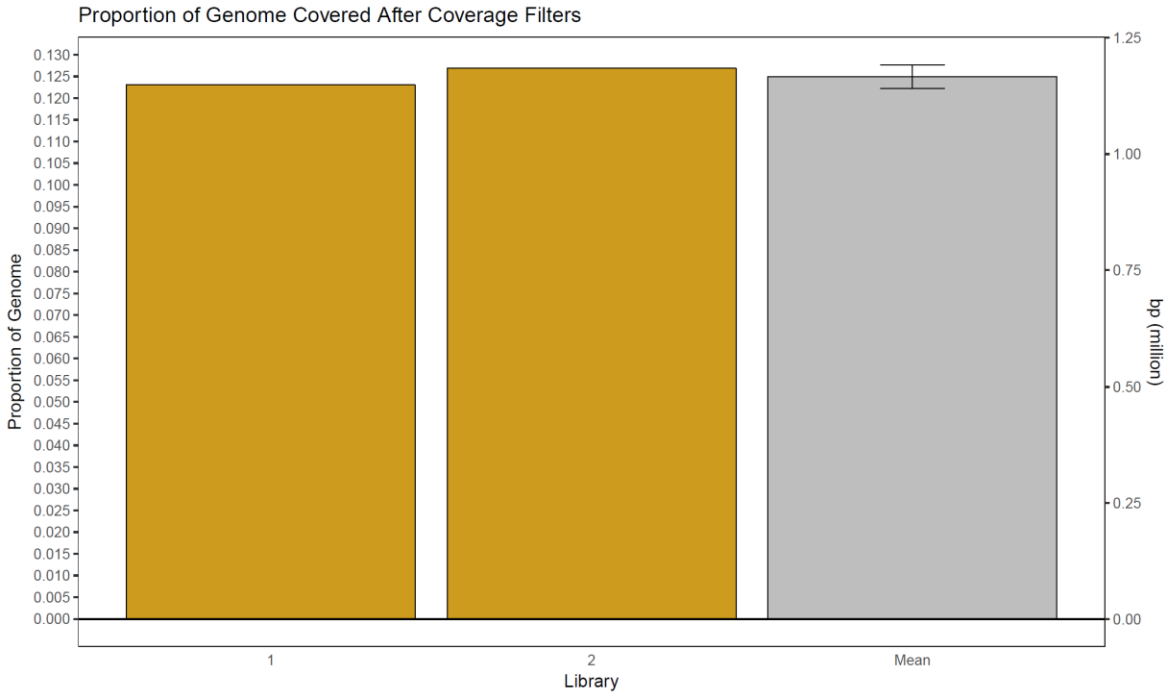
In the specified output folder, there will be a number of *txt* files and *pdfs*

The specified output file (*PP_stats.txt*) contains the raw statistics generated by *PPstats*. The first column of this file contains a unique string that is an identifier for the different analyses. The R script parses the file by the unique identifiers to make additional tables and plots. The string meanings are:

SUMMARY: summary information about the reference genome and number of populations
SCAFFOLD: information on scaffold length
COMB_TOT_BP: Total base pairs with sufficient read depth across all populations
COMB_TOT_PROP: Proportion of ref genome covered by reads with sufficient coverage
TMA: Total mean coverage (X) before filters of population
TMS: Standard deviation of coverage before filters of population
FMA: Filtered mean coverage (MINCOV > N < MAXCOV) of population
FMS: Standard deviation of mean coverage (MINCOV > N < MAXCOV) of population
FBC: Number of bp of ref assembly covered after filters (MINCOV > N < MAXCOV)
FPC: Proportion of ref assembly covered after filters (MINCOV > N < MAXCOV)
SCAF: Scaffold bp and proportions covered
ANC: Anchored bp and proportions covered
DOC/P: Proportion/bp of ref assembly covered at different minimum depths of coverage
CHRC/P: Proportion/bp of chromosomes covered after coverage filters

If run successfully, R will produce output plots and corresponding tables:

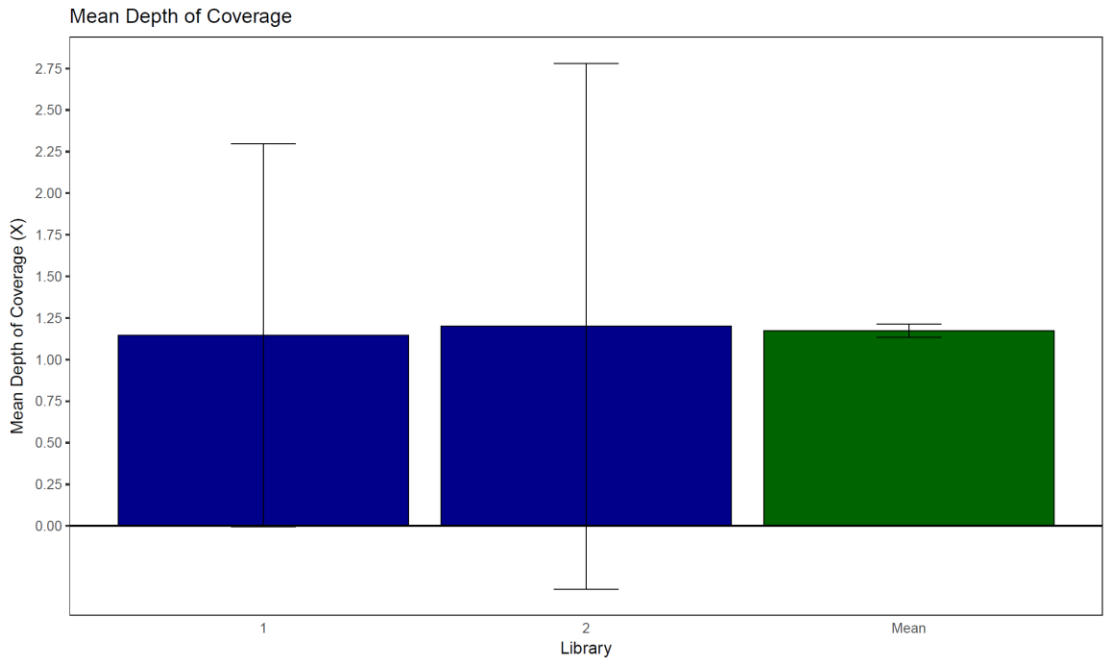
Take a look at *PP_stats_prop_cov.pdf*



After coverage filters, the example libraries cover about 12.5% of the reference genome. Mean shows the

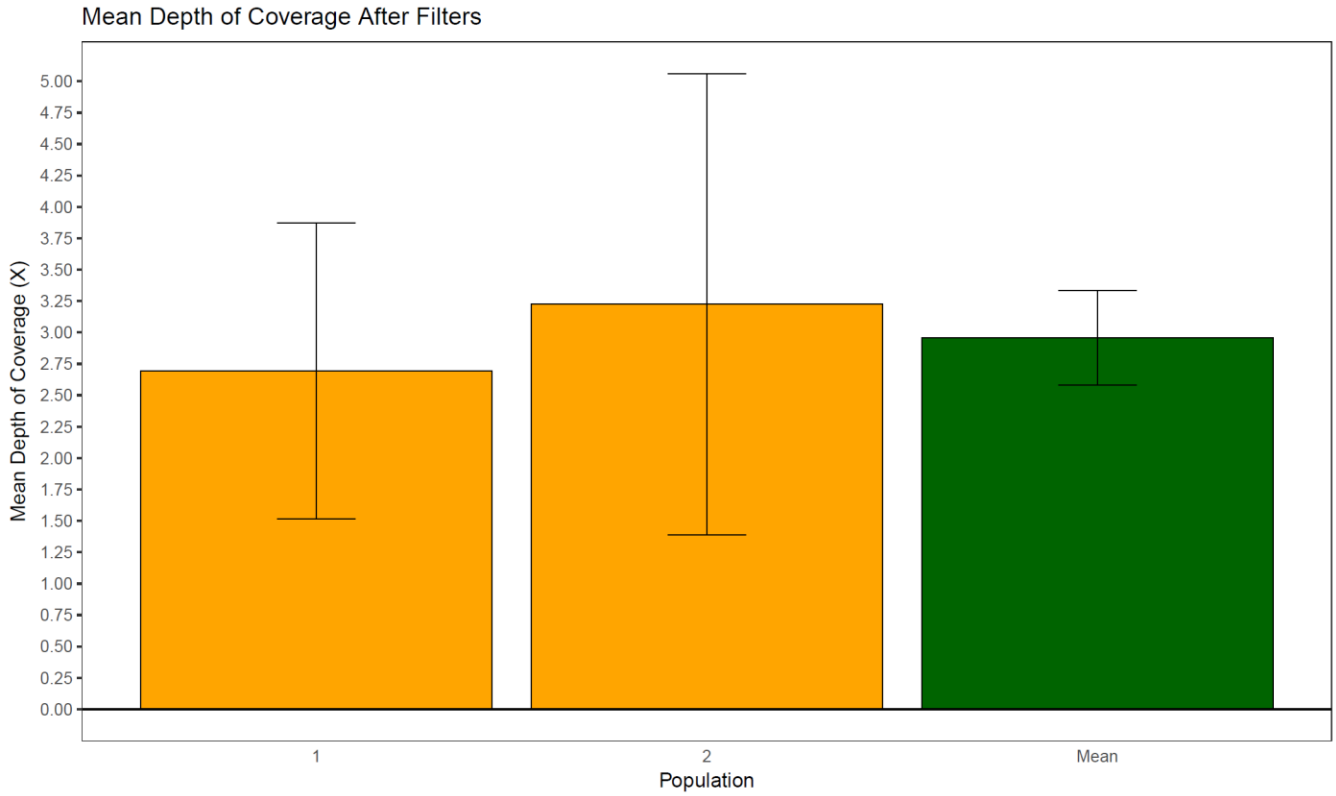
mean of the two libraries and standard deviation between the two.

Take a look at *PP_stats_mean_coverage.pdf*



Mean depth of coverage before coverage filters showing extremely low average coverage. Bars represent standard deviation for each library (1,2) and standard deviation between libraries (mean)

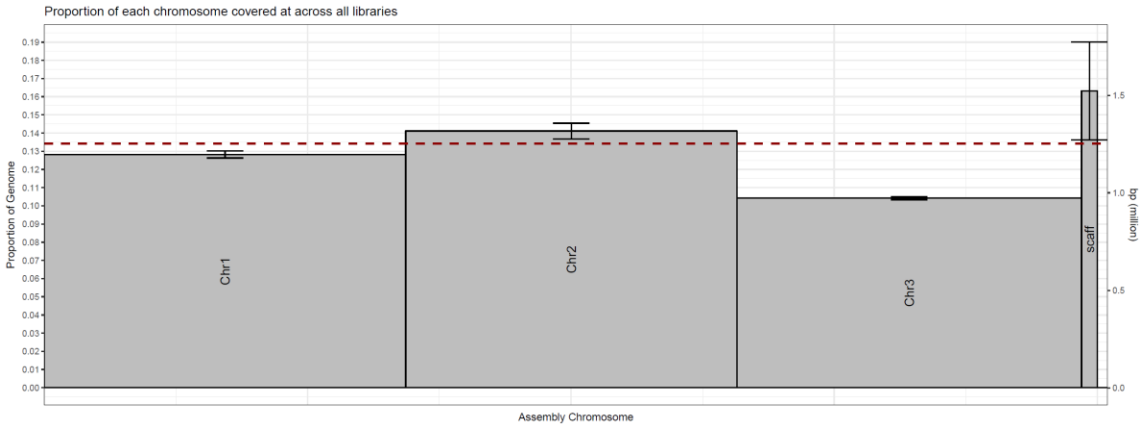
Take a look at *PP_stats_mean_filt_cov.pdf*



The mean coverage improves a bit when setting MINCOV=2.

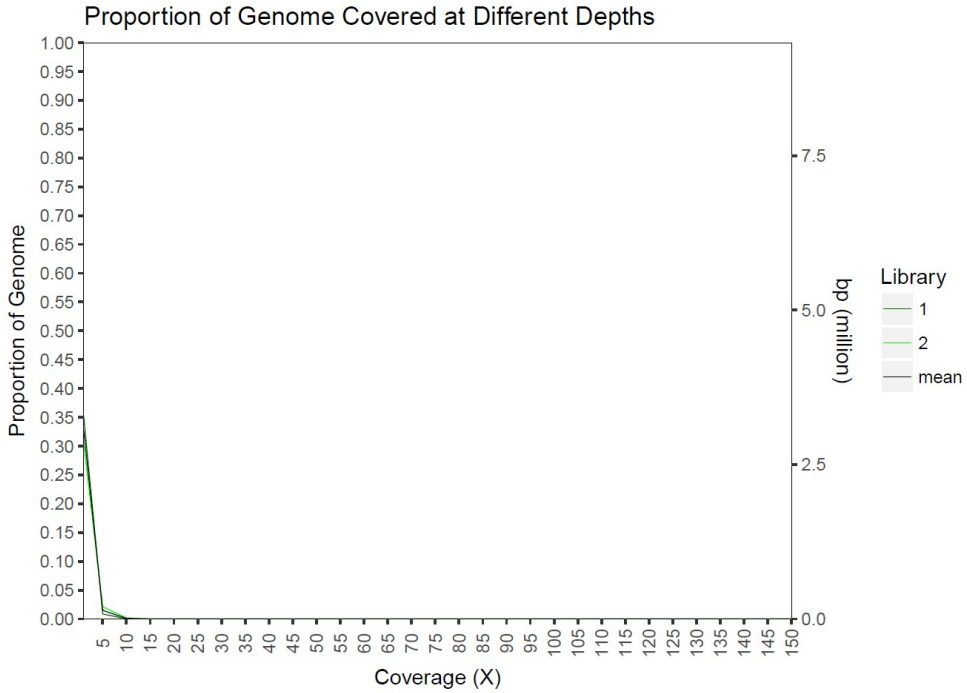
Take a look at *PP_stats_chr_prop_mean.pdf*

We can look at other plots that show chromosome alignment uniformity for each population, or anchored vs. unanchored plots, but this plot shows average uniformity across chromosomes and scaffolds. If there are too many chromosomes (>100) this plot will be skipped.



On average, after the MINCOV and MAXCOV filters, we can see that Chr3 is under-represented and the scaffold is over-represented (based on the dotted-red average line). The widths of bar correspond to the size of each chromosome / combined scaffolds.

Take a look at *PP_stats_prop_at_covs.pdf*



Though not impressive in our example1, this plot shows how much of the genome is covered at different depths of coverage. It helps determine coverage threshold selection and indicates how much data may be lost if minimum thresholds are increased. In our example, if we increased the minimum threshold to 5X we would only be covering about 2.5 % of the reference assembly.

In general, these plots are useful for determining how much of the reference assembly your reads are

covering and any potential biased alignment that could occur due to various library preparation protocols.

4) PPanalyze

PPanalyze is the final module of *PoolParty* and performs comparative analyses with the primary intention of discovering differentiated genomic regions between specified comparisons. *PPanalyze* produces output formats that can quickly be sorted or plotted with various tools.

Again, we can create a symbolic link to run *PPanalyze* from the command line

```
> ln -s ~/bin/poolparty/PPanalyze.sh /usr/local/bin/PPanalyze
```

a) Editing the config file.

The config file uses some of the previously generated files from *PPalign* to properly filter and analyze user- specified comparisons.

```
> cat pp_analyze.config

#Populations for analysis#
POPS=1,2
```

Specify the sample groups (actually columns from the *sync/fz* files*) that will be compared, separated by a comma. In our example, we only have two categories so there is only one comparison that can be made.

In other cases, you may have a sync file prepared by *PPalign* that has > 2 sample groups, and perhaps you want to look at differentiation between certain samples in this file. If we hypothetically had 5 sample groups (which could be a subset of all those that went through PPalign together), we can look at the average/global differentiation between all sample groups with POPS=1,2,3,4,5. Or perhaps we only want to know what is going on between sample groups 1 and 5, so: POPS=1,5.

Another possibility is that certain sample groups share a similar phenotype or other characteristic, and we wish to examine differences among them. Let's say sample groups 1-3 have phenotype "X" and sample groups 4-5 have phenotype "Z". We can look at differentiation between "X" and "Z" with the specification of: POPS=1:2:3,4:5. This comma (",") tells *PPanalyze* to average all divergence between sample groups 1,2,3 and 4,5 and subordinate (ignore) divergence within these supra-groups. If you have paired sampling, it may also be useful to apply Popoolation2's implementation of the Cochran–Mantel–Haenszel (CMH) test after running PPanalyze (explained below).

*It is critical to note that the input specified for PPanalyze is the column rank of samples (or 'sample groups' in the case of multiple libraries or individuals), not the names or numbers of the sample groups themselves (in the case only numeric characters were used to identify samples in the sample list, e.g. 01-N). Note also that PPalign will re-order populations from the sample list in alphanumeric order in the final *sync/fz* files (see above guidance on naming conventions for PPalign.)

```
#Input files and names#
PREFIX=phenoAB_analyze
```

A unique prefix that will be given to all files produced.

```
COVFILE=/userdir/phenoAB/filters/phenoAB_coverage.txt
```

The coverage file that was produced by *PPalign*, with path. This will be in the filters directory and is used to filter out SNPs by comparison-specific coverage.

```
SYNC=/userdir/phenoAB/phenoAB.sync
```

One of the sync files generated by *PPalign*, with path. If this was an individual analysis, you have the option of selecting the normalized sync file as well. Use *phenoAB.sync* for this example.

FZFILE=/userdir/phenoAB/phenoAB.fz

Frequency file generated by *PPalign*, with path. The full table is recommended, you may also choose the normalized frequency file if the individual normalization analysis was performed.

BLACKLIST=/userdir/phenoAB/filters/*phenoAB_poly_all.txt*

Optional blacklist, with path, that is simply one genomic position per line (CHR POS) of loci that you wish to remove. In this case we are just using sites that have >2 alleles in all populations as a blacklist; however, this list can encompass a combination of any SNPs you wish. For instance, if you found SNPs that are dominated by single individuals, or know of structural variant regions in the ref assembly that you wish to avoid, you can include those SNPs in this list.

OUTDIR=/userdir/phenoAB/filters/phenoAB_poly_all.txt

Output directory for files. Note that a temporary directory will be created in this directory as well, but will be deleted upon completion of the module

#Types of Analyses#

FST=on

Single-SNP FST analysis on filtered SNPs. There are two supported types of FST analysis (see below)

SLIDINGFST=on

Sliding-window FST analysis on filtered SNPs. Similar to the FST above, except averages a user- specified window.

FET=on

Fisher's exact test for differences in allele frequencies. Produces P-values for each SNP and can be a useful output for performing other windowed analyses such as Local Score. Note, to perform the FET analysis you must have a 2-tailed *perl* module installed.

NJTREE=on

Calculates SNP density at a user-specified window and created a neighbor-joining tree if >2 populations are present in the analysis.

#Global Parameters#

MINCOV=2

Minimum coverage needed to retain a SNP. MINCOV=2 means that both pop_1 and pop_2 must have a minimum coverage > 2. The more populations that are included in an analysis, the more SNPs will be lost due to minimum coverage not being met across all populations. Set to a threshold of 2 for this example.

MAXCOV=100

Maximum coverage needed to retain a SNP. Serves generally as a paralog filter, yet the value should depend on the organism and coverage from *PPstats*. For instance, it makes sense to base this value on the mean coverage of all reads, and set it to a set number of standard deviations above the mean. If paralogs or duplication is not an issue in your organism, this value can be set very high. For this example, with very low mean coverage, leave at 100.

MAF=0.06

Analysis-specific minor allele frequency. Though global MAF filters have already been applied, this value is based solely on user-specified comparisons. In our example, we have already filtered out MAF < 0.05 between pop_1 and pop_2, so let's set MAF to a

high value to filter SNPs further.

#FST Parameters#

FSTTYPE=traditional

Type of FST analysis to perform for both FST and SLIDINGFST. The two options are traditional or an equation by karlsson (Karlsson *et al.* 2007). The karlsson method considers allele counts which makes it appropriate for pooled data. Stick with traditional for the example.

WINDOW=10000

Window size in bp for FST window analysis. This will be the window size, according to the ref genome, in which FST is averaged.

STEP=5000

Step size for SLIDINGFST. Must be smaller than window size.

NIND=5

Total number of individuals for FST analyses. Used for sample size correction.

#NJ tree Parameters#

STRWINDOW=10000

The window size to look for SNP density and in which allele frequencies are combined. If set to 1, will not calculate SNP density and will not use a combination method for allele frequencies.

BSTRAP=2000

Number of bootstraps to perform on neighbor-joining (NJ) tree. The NJ tree uses Nei's genetic distance and will produce a consensus tree with node support as well as a single tree with node support. The NJ tree will not be produced for the example since there are less than three populations.

AFFILT=1

Allele frequency filter for NJ trees. This is a crude method to attempt to remove adaptive regions of the genome that may overpower neutral structure. In general, should be set to 1, which means do not filter allele frequencies, or filter allele frequencies greater than 1 (which is impossible).

METHOD="mean"

Method to use when investigating SNPs within a window size for the NJ tree. If SNP density is not uniform due to biased alignment, then certain regions of the genome may contribute more to identifying population structure. The windowed approach in NJ tree attempts to dampen these effects by looking at allele frequencies of SNPs within a window and either taking the mean ("mean"), choosing the SNPs with least deviation in the window ("sdmin"), the most deviation in the window ("sdmax"), a random SNP in the window ("random"), the SNP with the largest allele frequency difference ("rangemax"), the SNP with the smallest allele frequency difference in ("rangemin"), or the first SNP of the window ("first").

#Dependencies#

POOL2=/usr/local/bin/popoolation2_1201/

The *Popoolation2* directory is the only dependency for this package. Be sure to modify this to provide the correct path (same as in the PPalign *config* file).

b) Running PPanalyze

Run *PPanalyze* similar to other modules:

```
> PPanalyze pp_analyze.config |& tee log.analyze_example.txt
```

The run should take < 5 min to complete. Similar to *PPalign*, alerts are printed that indicate the number of SNPs removed from filters.

An important ALERT here is "ALERT: Pops 1 2 are now in the order of 1 2 in the subset sync file." For the example, the order of populations has not changed; however, if a subset of populations is specified, and done so in a different order, the resulting sync file will be in that order. This line thus indicates which order the populations are in after sub-setting (this will be critical if running the Cochran–Mantel–Haenszel (CMH) test script).

c) Output files

Within the output files are raw Popoolation2 outputs ("_raw"), and outputs formatted for plotting ("_analysis").

```
> awk '$4 > 0.75' phenoAB_analyze.fst | wc -l
```

We see that 137 SNPs had very high FST (>0.75)

```
> awk '$4 < 0.05' phenoAB_analyze.fet | wc -l
```

Only 83 SNPs had significant allele frequency differences (p <0.05), likely due to the low coverage allowed in the example.

PPanalyze creates subset *sync* and *fz* files for additional analyses. For instance, PPruncmh.sh uses *Popoolation2* to perform a Cochran–Mantel–Haenszel (CMH) test on subset sync files to look for consistent changes in allele frequencies between biological replicates. If you were to have sampling from different locations: phenotype A from location 1, phenotype B from location 1, phenotype A from location 2, and phenotype B from location 2, you can identify consistent deviations in allele frequencies between phenotype A and B from different geographic localities.

d) PPmanhat.sh: Plotting results

There are numerous ways to visualize results. Provided with *PoolParty* is the PPmanhat.sh utility module, which uses *qqman* in R (Turner 2014). Other python-based (matplotlib) or perl-based (e.g., Circos; Krzywinski *et al.* 2009) packages could work as well.

PPmanhat.sh does not require a config file. If desired, create a symbolic link as with the other shell scripts:

```
> ln -s /downloads/poolparty/PPmanhat.sh /usr/local/bin/PPmanhat
```

Initiating PPmanhat alone, or with the -h option will bring up help in the terminal:

```
> PPmanhat -h
```

```
PPManhat : Uses qqman to make Manhattan plots (.pdf, .png) of results from PPanalyze

(FST,FET,SFST) Usage:    PPmanhat -i [input] -o [output name] -c [chromosome range file]

-l [-log 10 trans]
                -a [analysis type] -s [scaffold heading] -r [scale by value]
                -C [plot chromosome] -R [plot range within chromosome] -1 [color1] -2 [color2]
                -L [threshold line] -t [plot type] -p [make pdf too]
 Argument                          Description                                                    Default

 - i   [input]                     : Input file with four column format (CHR,POS,SNP,FST/P)       [REQUIRED]
 - o   [output name]               : Prefix used to define the analysis                           [REQUIRED]
 - c   [chromosome range file]     : File with chr names and starting/ending positions (CHRbp.txt) [NULL]
 - l   [-log 10 trans]             : Perform -log10 transform on column4                          [FALSE]
 - a   [analysis type]             : Descriptor for Y axis (i.e., FST,-log10)                     [Differentiation]
 - s   [scaffold heading]          : String identifier for scaffolds, if present                  [scaffold]
 - r   [scale by value]            : If TRUE, Y range is scaled. If FALSE, Y range is 0-1         [TRUE]
 - C   [plot chromosome]           : Plot a single chromosome (integer)                           [NULL]
 - R   [plot range within chromosome] : Plot bp range within -C (two integers, comma-separated)   [NULL]
 - 1   [color1]                    : Color of odd chromosomes (R base colors)                     [darkred]
 - 2   [color2]                    : Color of even chromosomes (R base colors)                    [darkblue]
 - L   [threshold]                 : Draw threshold line at this value (integer)                  [NULL]
 - t   [plot type]                 : How to plot points (point,line,bar)                          [point]
 - p   [make pdf]                  : Make a pdf output in addition to png output                  [FALSE]
```

PPmanhat can directly plot a number of *PoolParty* outputs such as FST, FET, SFST, CMH, FLK, LocalScore, and SNP-density outputs. We provide a script to run standard analysis plots for all chromosomes, *PPrun_analysis_plots.sh*, from within the analysis directory (see Example2). There are also many options to customize your Manhattan plot as shown below, but only two required arguments: -i (input file) and -o (output prefix).

In each case, the PPmanhat input is in a 4-column format:

Col. 1) Chromosome, Col. 2) Position, Col. 3) SNP-ID, Col. 4) Differentiation value.

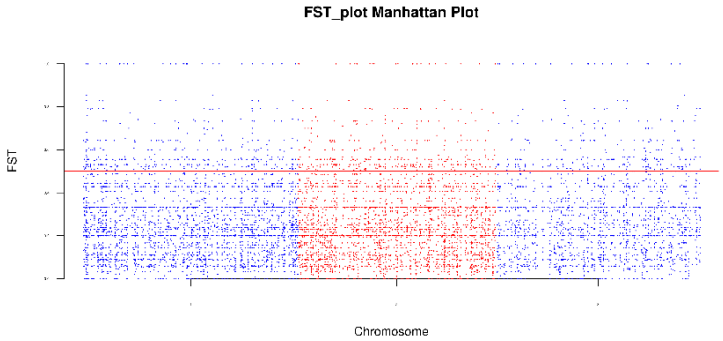
The SNP-ID is a consecutive number from 1 to *N* SNPs, whereas the differentiation value depends on the analysis (FST, p-value, -log10p, *etc*.).

Let's plot a couple PPanalyze results to understand additional arguments:

Plot basic FST:

```
> PPmanhat -i phenoAB_analyze.fst -o FST_plot -a FST -s Scaff -1 blue -2 red -L 0.5
```

Take a look at FST_plot.png



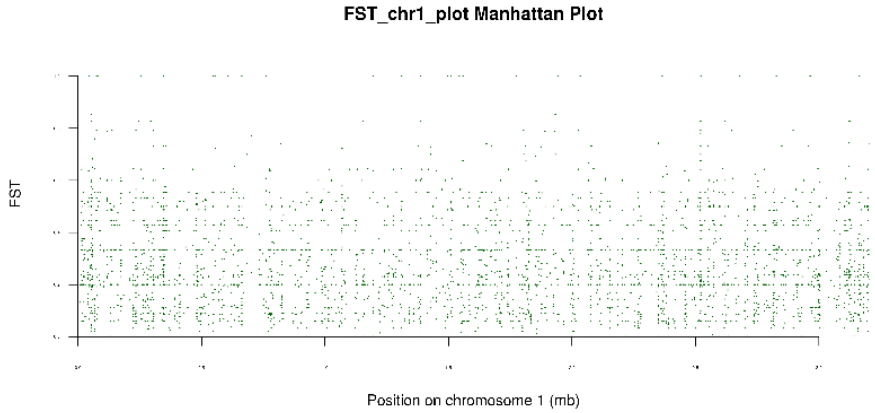
-a FST, defines the y axis label.
-1 blue -2 red, sets alternating colors. These colors use R's color scheme and naming.
-L 0.5, draws a threshold line at y value.
-s Scaff removes the scaffolds. Usually, scaffolds cannot be effectively plotted when in high numbers.

This ugly plot reminds us that bad data equals bad data out. For these ~13,000 SNPs, the plotting only takes about 1-2 seconds. However, when SNP ranges are in the millions, you can expect the plotting to take more time.

Perhaps we just want to plot a single chromosome:

```
> PPmanhat -i phenoAB_analyze.fst -o FST_chr1_plot -a FST -s Scaff -1 darkgreen -C 1
```

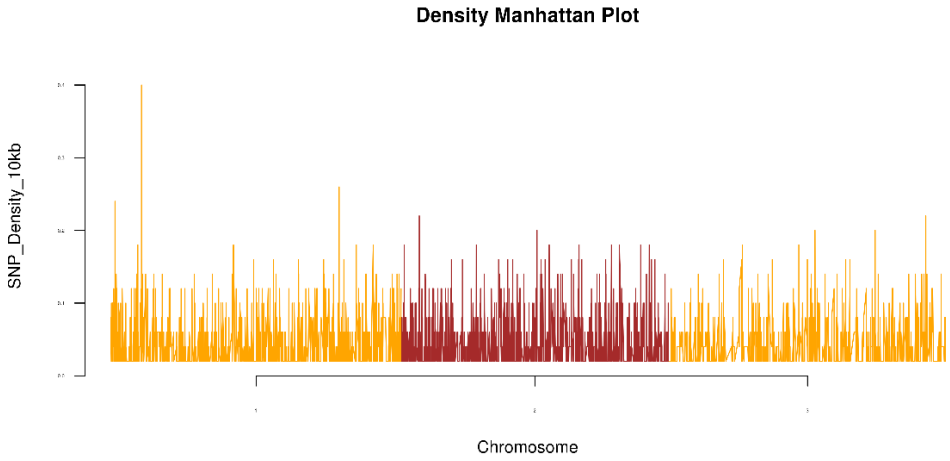
Take a look at FST_chr1_plot.png



We can use a similar method to plot SNP density across the PP reference genome. This time, a line chart might be more effective:

```
> PPmanhat -i phenoAB_analyze_density.txt -o Density -a SNP_Density_10kb -s Scaff \
-1 orange -2 brown -t line
```

Take a look at Density.png



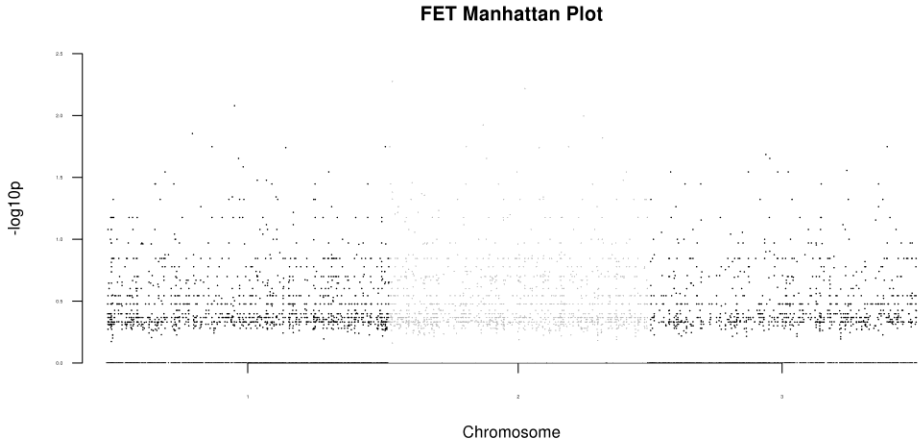
This plot may be useful to identify SNP-dense regions as seen in the beginning of chromosome 1.

Finally, let's plot Fisher's test results (-log10[p]) and use -c, the chromosome range file. This file, called phenoAB_CHRbp.txt, can be found in the OUTDIR. This file defines the start and end ranges of each chromosome so that the plotting extent is the actual genome size, not the range of the SNPs. For instance, if a chromosome is 1 mb long, and the first identified SNP is at 50 kb, and the last is at 200 kb, then without this file the chromosome plotting range would be 50 kb to 200 kb. With this file, the plotting range would be 1 to 1mb.

First, let's move phenoAB_CHRbp.txt into the analysis folder so we don't have to specify the directory. Then:

```
> PPmanhat -i phenoAB_analyze.fet -o FET -a -log10p -s Scaff -1 black -2 gray \
-c phenoAB_CHRbp.txt
```

Take a look at FET.png



The extent of the chromosomes now matches the true start and stop positions. This result is often hard to see if SNPs are evenly distributed across the genome.

5) Advanced runs and utilities

PoolParty comes with multiple utility and analysis scripts for achieving more complex tasks. To illustrate these, let's re-run PPalign and PPanalyze with a modified samplelist. Make the changes below to samplelist.txt in the input directory.

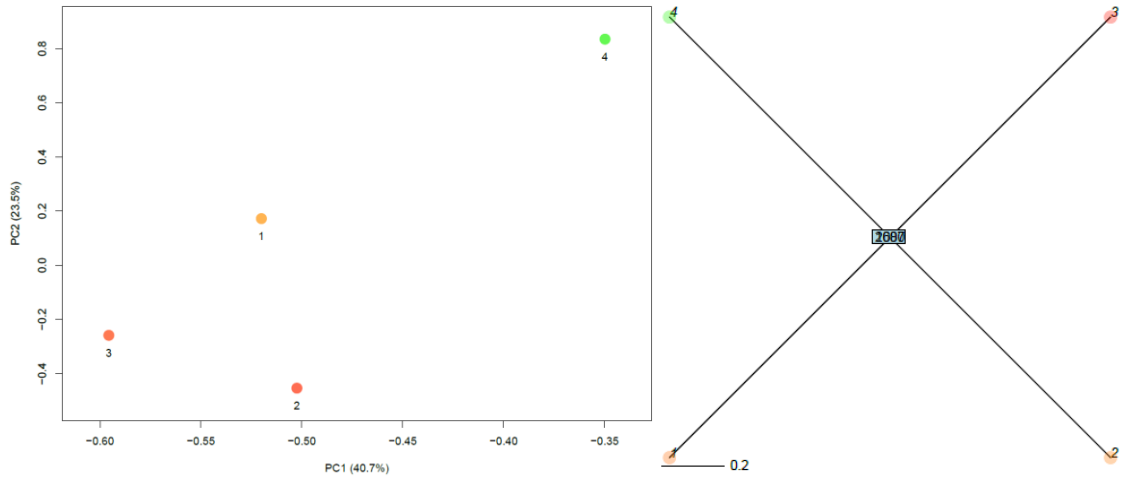
Change the library grouping designations in samplelist.txt as follows:

```
P101_103_R1_PopA.fq.gz       1
P101_103_R2_PopA.fq.gz       1
P102_97_R1_PopA.fq.gz        2
P102_97_R2_PopA.fq.gz        2
P201_104_R1_PopB.fq.gz       3
P201_104_R2_PopB.fq.gz       3
P202_103_R1_PopB.fq.gz       4
P202_103_R2_PopB.fq.gz       4
P203_97_R1_PopB.fq.gz        4
P203_97_R2_PopB.fq.gz        4
```

This modified samplelist is now indicating there are 4 populations, or categories, which allows us to utilize the bulk of *PoolParty* scripts. Run PPalign and PPanalyze with different directories and output names and feel free to change parameters - although pops in PPanalyze should be set to 1,2,3,4.

Running the same parameters as before, now with 4 categories, we get an interpretable PCA and NJ tree from PPanalyze.

Take a look at output_PCA.pdf
Take a look at output_consensus.pdf



Not much structure here, although the PCA indicates that 4 may be different from the other three.

a) CMH test

If we pretend that our data are biologically meaningful and we included replicates from each source (e.g. population) across phenotypes, we can use PPruncmh to determine consistent allele frequencies between phenotypes using the Cochran–Mantel–Haenszel (CMH) test. Users should read guidance on CMH study design carefully to ensure their testing design is valid (i.e., it is intended for tests among replicate pairs of samples/sample groups, but pairwise tests among independent groups are not valid). To achieve this, we will pretend:

Sample grouping 1 is from location A with phenotype X
Sample grouping 2 is from location B with phenotype Y
Sample grouping 3 is from location B with phenotype X
Sample grouping 4 is from location A with phenotype Y

We will then use the CMH test to determine regions of the genome that have consistent allele frequency differences between 1-4 and 2-3. This is essentially a control method for population structure. It is important to note that these numeric designations refer to 1) sample groupings if individuals (or multiple libraries) are utilized with PPalign and 2) **column order** in the subset *sync*/*fz* file. If solely numeric designators were used in the sample file for PPalign (e.g. 01-N), be aware that these are NOT what are being specified here (and that PPalign may have re-ordered these in the output; see above notes on sample specification for PPalign). Review the log output from PPanalyze to determine which order sample groupings are in now in those files, as PPanalyze also re-orders populations to group first by any phenotype (i.e. before and after the ","), and then by alphanumeric order (following PPalign). E.g., the above structure, where we wish to test across **PHENOTYPE**, would have been specified in PPanalyze as

1:3,2:4

Since samples 1 and 3 vs. 2 and 4 have the same PHENOTYPE. And we can expect the following line in the PPanalyze output:

Pops 1 3 2 4 are now in the order of 1 2 3 4 in the subset sync file

Thus, we need to tell the CMH test which columns represent samples from the same genomic background (e.g. population), in this case what are now 1+4 (1 and 4 from location A, respectively) and 2+3 (3 and 2 from location B, respectively). This is specified (using the -p flag on command line) as

1-4,2-3

which tells the CMH test to look for consistent *differences* between the frequencies in columns 1 (sample 1) and 4 (sample 4) as well as 2 (sample 3) and 3 (sample 2). The PPrun_analysis_plots.sh script will optionally run the CMH test, with column specification as input, as well as a local score test on CMH output (see below). Or, after making a symbolic link, we can run PPruncmh alone or with -h to see help:

```
> PPruncmh -h
```

```
Usage: PPruncmh -i [input] -o [outname]
                -p [pops] -P [pooldir]

Argument            Description                                                    Default

 - i    [input]    : Input sync file                                                [REQUIRED]
 - o    [outname]  : Output prefix name                                             [REQUIRED]
 - p    [pops]     : pops to compare. i.e., 1-2,3-4 where 1&2 are from same locality [REQUIRED]
 - P    [pooldir]  : Popoolation2 base directory                                    [REQUIRED]
 - m    [mincov]   : Minimum coverage override for unanalyzed sync file            [1]
 - x    [maxcov]   : Maximum coverage override for unanalyzed sync file            [10000]
```

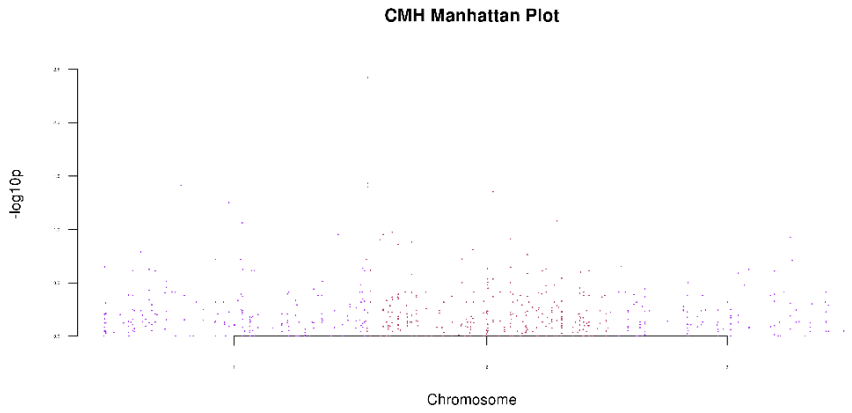
Since Popoolation2 is doing the heavy lifting, we must specify its base directory. The input file will be the analyzed sync file (which is already filtered by align and analyze parameters).

```
> PPruncmh -i ouput_analyze.sync -o CMH_out -p 1-4,2-3 -P /usr/local/bin/popoolation2_1201
```

Both CMH_out_raw.cmh and CMH_out.cmh are produced. Raw is the Popoolation 2 output, whereas the other is a standard 4-column output for use with PPmanhat. We can plot CMH_out.cmh using PPmanhat and will use the -l argument to transform the p-values into -log10(p).

```
> PPmanhat -i CMH_out.cmh -o CMH -a -log10p -s Scaff -1 purple -2 maroon -l TRUE
```

Take a look at CMH.png



The greatly reduced number of SNPs is due to breaking the libraries into lower-coverage populations. With this, there are some outliers (like on chromosome 2), but nothing significant after Bonferroni correction (alpha=0.05).

b) FLK test

FLK (Bonhomme *et al.* 2010) uses an allele frequency input and a phylogenetic structure correction to identify SNP outliers. Due to the nature of its input, it is an ideal outlier test for pool-seq data. Consequently, PPrunflk serves as an efficient method to run FLK on PP allele frequency tables (make symbolic link to script, or run via PPrun_analysis_plots.sh).

```
> PPrunflk -h
```

```
PPrunflk : Runs FLK script using an allele frequency

table (.fz) Usage:       PPrunflk -i [input] -o

[output name] -g [outgroup] ]

 Argument                          Description                                                    Default

 - i   [input]                     : Input file in PP .fz format                                  [REQUIRED]
 - o   [output name]               : Prefix for output name                                       [REQUIRED]
 - g   [outgroup]                  : Specify an outgroup if desired (population integer)          [FALSE]
```

PPrunflk only requires a frequency table input (produced by PPanalyze), an output name, and an outgroup for rooting the tree (which is the population number in the fz table), if desired. Let's run FLK and assume that population 4 is an outgroup. If we do not specify the -g argument, midpoint rooting will be used.

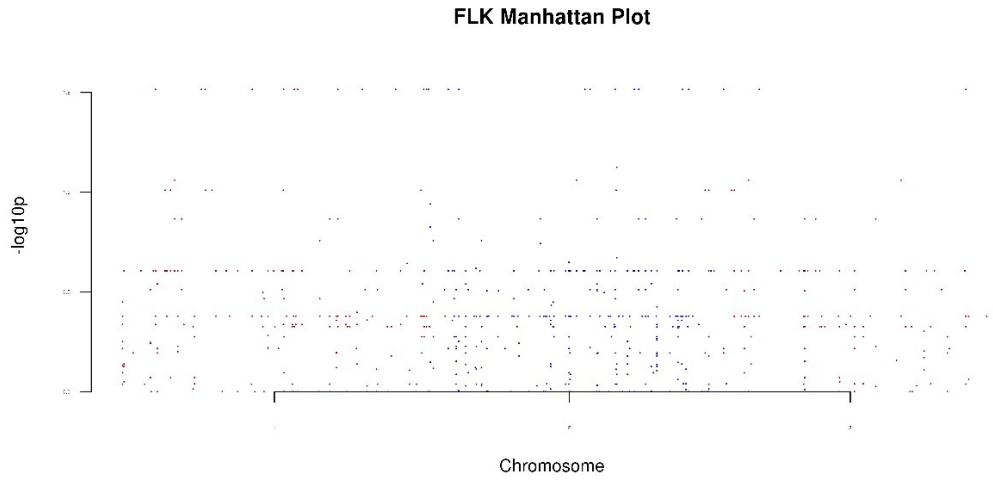
```
> PPrunflk -i outname_analyze.fz -o FLK_run -g 4
```

A detailed table with likelihoods, p-values, and q-values is produced (FLK_run_flk_results.txt) as well as a simple 4-column output with FLK-pvalues (FLK_run.flk).

We can graph FLK_run.flk with PPmanhat:

```
> PPmanhat -i FLK_run.flk -o FLK -a -log10p -s Scaff -l TRUE
```

Take a look at FLK.png



More messy data with no significant values after multiple-comparison corrections.

c) Local Score test

PPrunls will run the Local Score analysis (Fariello *et al.* 2017) on a set of p-values in a 4-column format, which means it is useful for running on the four-column output of the Fisher's Exact Test (*.fet*) or CMH (*.cmh*), which the PPrun_analysis_plots.sh script will do automatically (the latter if CMH is selected). This test can determine regions of the genome that are differentiated while considering linkage equilibrium; however, itis strongly recommended to read Fariello *et al.* 2017 prior to running this analysis and reading their supplementary material.

```
> PPrunls
```

```
PPrunls : Uses local score R script to calculate significant regions from p-values

(FET,CMH,FLK) Requires PPanalyze output format: 4 columns (Chr,pos,SnpID,p)
Please read Accounting for linkage disequilibrium in genome scans for selection without individual
genotypes: The local score approach (Fariello et al. 2017)

 Usage:   PPrunls -i [input] -o [output name] -s [scaffold prefix] -x [tuning parameter]

 Argument                          Description                                                    Default

 - i   [input]                     : Input file with four column format (CHR,POS,SNP,P)           [REQUIRED]
 - o   [output name]               : Prefix for output name to define the analysis                [REQUIRED]
 - s   [scaffold prefix]           : If scaffolds present, prefix that identifies them            [scaff]
 - x   [tuning parameter]          : Override tuning parameter for local score (number)           [NULL]
 - u   [force uniformity]          : Force p-value distribution to be recognized as uniform       [NULL]
```

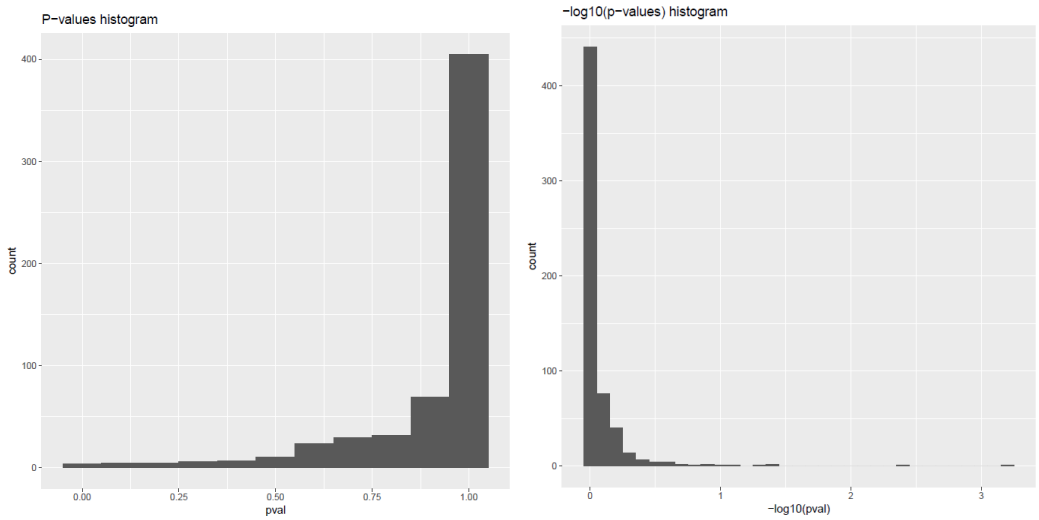
PPrunls has built in functions to automate the local score process. For instance, it will determine if p-value distributions are non-uniform and auto-tune the xi parameter based on the distribution of -log10pvalues and mean score. It is imperative that distribution outputs are observed and values are well understood before using this technique.

Let's use FET p-values to run ls. We will force uniformity due to the reduced number and density of SNPs.

```
> PPrunls -i phenoAB_analyze.fet -o LS -s Scaff -u TRUE
```

LS_pvaldist.pdf and LS_blog10.pdf are the distributions of pvalues and -log10(p), respectively:

Take a look at LS_pvaldist.pdf and LS_blog10.pdf



The distributions are far from uniform which will generally be true in a dataset where evolutionary forces are acting. If uniformity is not forced, and a user does not select xi, PPrunls will choose an xi based on quantiles.

In the above run, LS_mean_sig.txt and LS_chr_sig.txt are produced, which contain the significant thresholds based on different alpha levels (analogous to a Bonferroni correction). Thresholds will be different for each chromosome as well, which is shown in LS_chr_sig.txt.

The other important output is this *sig.txt. These are genomic regions that have been deemed significant at different alpha levels.

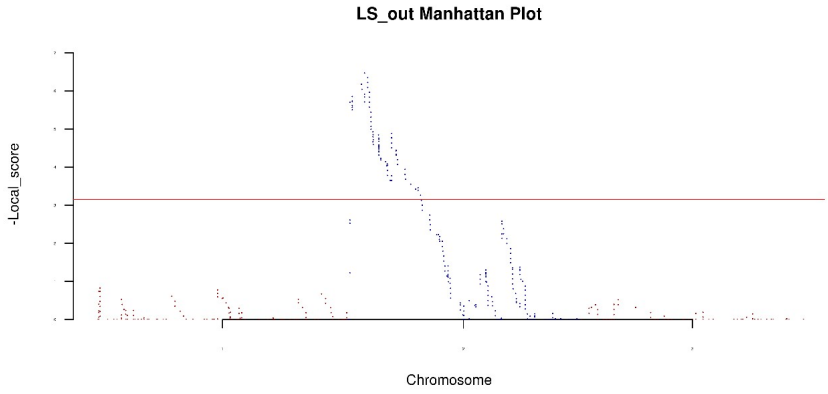
> cat LS_05sig.txt

```
chr         beg         end         peak
Chr2        40604       224832      6.471292
```

This illustrates that the analysis determined a significant region between 40604 and 224832 on chromosome 2 with a maximum local score of 6.47. While the local score on the y axis is analogous to -log10(p), its range can vary based on differentiation and the tuning parameter (xi).

Finally, we get LS.ls, which is the 4-column format for plotting. Let's plot this with the average chromosome significance at 0.05 in LS_mean_sig.txt:

```
> PPmanhat -i LS.ls -o LS_out -a -Local_score -s Scaff -L 3.15
```



Forcing uniformity will lead to high differentiation and false-positives if the p-values are not evenly distributed. Thus, selection of the tuning parameter (xi) is essential in empirical datasets.

Sub-setting analyses

PPsubset is an effective way to further subset PPanalyze files. This utility can further apply coverage thresholds, or subset the analyses by specific population comparisons (symbolic link!):

```
> PPsubset -h

Subset PPanalyze results by various thresholds

Usage: PPsubset -t [analtype] -a [analname] -o [output]
-m [mincov] -x [maxcov] -p [propsnps] -n [numsnps] Argument Description      Default
t           [analtype]  : either FST, SFST, or FET[REQUIRED]
a           [analname]  : prefix used to define the analysis      [REQUIRED]
d           [directory] : location of analyzed files  [CURRENTDIR]
o           [outname]   : Id suffix of reclassed files            [rc]
m           [mincov]    : minimum coverage      [1]
x           [maxcov]    : maximum coverage      [10000]
P           [propsnps]  : minimum proportion of SNPs in window[0]
n           [numsnps]   : minimum number of SNPs in window            [1]
p           [pops]      : reclassification of populations         [NULL]

        If > 2 populations, script requires analname_avoidcols.txt Pop classification uses same format as
PPanalyze
```

PPsubset requires the prefix name for the analysis that was performed. For instance, let's say we only wanted to observe FST between populations 2 and 4, and raise the minimum coverage threshold to 5 x.

```
> PPsubset -t FST -a output -m 5 -p 2:4 -o Reclass
```

We now get output_Reclass.fst, which shows FST between populations 2 and 4 with a higher minimum coverage threshold of 5X.

s

6) Beyond *PoolParty*

Depending on the question and genomic resources available, there are many branching paths of analysis after *PoolParty*. For instance, if gene annotation is available, it may be useful to identify which genes highly differentiated SNPs represent. Enrichment of GO categories is also a common practice. Even if annotations are not available, running BLAST on sequences around genes of interest may give insight to the genomic basis of traits of interest.

Some *PoolParty* outputs, such as FET tests, can be used for more additional statistical analyses of differentiation that control for noise or false-positives such as GLMS (Wiberg *et al*. 2017). Depending on the structure of libraries, additional outlier tests that control for population structure may help differentiate adaptive from neutral variation.

7) References

Bonhomme, M., Chevalet, C., Servin, B., Boitard, S., Abdallah, J., Blott, S., & SanCristobal, M. (2010). Detecting selection in population trees: the Lewontin and Krakauer test extended. *Genetics*, 186(1), 241-262.

Fariello, M. I., Boitard, S., Mercier, S., Robelin, D., Faraut, T., Arnould, C., ... & Gourichon, D. (2017). Accounting for Linkage Disequilibrium in genome scans for selection without individual genotypes: the local score approach. *Molecular Ecology*.

Karlsson, E. K., Baranowska, I., Wade, C. M., Hillbertz, N. H. S., Zody, M. C., Anderson, N., ... & Comstock, K. E. (2007). Efficient mapping of mendelian traits in dogs through genome-wide association. Nature genetics, 39(11), 1321.

Krzywinski, M., Schein, J., Birol, I., Connors, J., Gascoyne, R., Horsman, D., ... & Marra, M. A. (2009). Circos: an information aesthetic for comparative genomics. *Genome research*, *19*(9), 1639-1645.

Turner, S.D. qqman: an R package for visualizing GWAS results using Q-Q and manhattan plots. biorXiv DOI: 10.1101/005165 (2014).

Wiberg, R. A. W., Gaggiotti, O. E., Morrissey, M. B., & Ritchie, M. G. (2017). Identifying consistent allele frequency differences in studies of stratified populations. *Methods in Ecology and Evolution*.