WORKFLOW

Download .fastq.gz → unzip → clip adapters → look @ quality/ decide where to trim → trim → quality filter → assemble/ align to genome → GATK to call SNPs/quality filter → data analysis!

Double check that your sequencing company trimmed the barcodes off of the ends.

In all awk scripts, you will get a syntax error if there are "smart quotes" or m dashes (like Word produces automatically) rather than straight, unformatted quotes or n dashes. Sometimes this means you have to edit by hand if you are copying and pasting from a Word document. It will save a lot of headaches to turn off auto-correct for these components.

At any point, you can check how much space you are taking up on a drive with `$ du -hs /directory/`

Useful tools and tricks:

- ctrl u saves what you've typed, ctrl y pastes it
- `$ find /some/path –name somefile`
- `$ ls -1 | wc –l` tells you how many files are there, including directories. If you just want to count files of a certain type, change the ls command to look for *.txt or whatever you want

- Install packages with wget package_name or yum install package_name
- First you have to do QC to figure out where to trim. Running FastQC will produce an html file showing summary statistics about the quality, and you can see at what point in the read the quality drops off into the red. Look through each read of each sample and decide where you are going to trim each read; record this in a file (how many reads, what you keep, what you trim). For some programs, the reads have to be the same length for all samples, but for this pipeline they can be different lengths.
- Log in: `$ ssh xyz@123.45.…` and supply the pw when prompted
- If you have ancient samples or really short (<290 bp) sequences, you should merge now, before trimming or clipping the adapters (prinseq can be used for merging + quality filtering). If modern samples, CH and MH do not merge at all because the sequences are often too long for the program to find the overlap, which will result in a lot of unmerged reads (I had about 60% merge successfully, which they say is a lot). Then you can map the merged reads AND the unassembled reads in bwa or whatever aligner you use, and then use samtools merge.
- Clip adapters if the sequencing facility did not do so! You should be able to use regular expressions to remove all adapter/index sequences by typing the common part first and then using RegEx
  - To find out if there are adapter sequences, search for the occurrence of the common part in the fastq files: `for i in *.fastq.gz; do gunzip -c $i | grep -c -A 2 -B 1 GGCCTCTTGATCA; done` where -A means print x number of lines after the hit and -B means print x number of lines before the hit
  - `fastx_clipper –a GATCGGAA… -l 15 \ -n –i myseqs.fastq –o myseqs.clipped.fastq`
- Merge Read 1 and Read 2 only if you have ancient/ degraded samples; this will help with mapping.
  - Use PEAR; only 1% more reads with p=0.05 than p=0.01 (default). Disable with p=1.0; this assembles an additional ~4% of reads. My ancient samples didn't merge well (they 'merged' end-to-end), so I am changing the minimum overlap (-v flag) to 20 bp and adding the -m flag which sets a max sequence length (151 + 16 + wiggle)
  - `$ (for i in *_1.fastq; do /home/genetics/pear-0.9.0-src/src/pear -p 0.05 -f $i -r ${i/1.fastq/2.fastq} -v 20 -m 175 -o ${i%1.fastq}; done &)`
- Trim reads
  - Run fastqc on all samples—generates .fastqc files from .fastq with the command `$ ./FastQC/fastqc fastq_data.fastq` (http://www.bioinformatics.babraham.ac.uk/projects/fastqc/)

  - To view on home computer, download a fastqc html file locally: `scp xyz@123…456:/fqc/*.html ./`
  - Trim reads based on fastqc results (I make subdirectories for samples with differing quality so I can trim them to different lengths): `$ for i in *_R1_001.fastq; do python fastq_trimmer.py $i; done` (can Google "Illumina quality coding" to figure out what quality each letter corresponds to)
  - To remove 10 bases from the end you write x=line[:-11] (counts include 0)
  - Makes a .fastqNEW file that you have to rename to .fastq so you can process them downstream. `for i in *.fastqNEW; do mv $i ${i%fastqNEW}trim.fastq; done`
  - Rename is something like `$ for i in *.fastq; do var='echo $i|awk -F '_' '{print $2 "_R1_trim.fastq"}''; mv $i $var; done`, where the 'print $2' part tells it to print what's before the 2nd underscore. The following will give you multiple parts of the name: `for i in *fastqNEW; do var='echo $i|awk -F '_' '{print $1 "_" $2 "_" $3 "_" $4 "_R2_trim.fastq"}''; mv $i $var; done`
- Compress original fastq files to save space
  - `$ tar –cvzf compressed_name.tar.gz file1 file2 file3`
- If samples were sequenced on multiple runs or lanes, concatenate them (sooner is better than later; I do it after trimming or quality filtering). If you will merge reads, make sure the concatenation is in the same order for both reads. If the replicates are in different folders, enter into one of the folders and concatenate with a loop: `for i in *fastq; do cat $i ../Mayrun/$i > ../allAxeq/$i; done`
- Quality filtering
  - Quality filter trimmed reads (Martin Kircher's QualityFilterFastQ.py or fastq_quality_filter from the FastX toolkit (TC ran the latter with modern DNA, which is probably better)
  - For some filters you'll need to input criteria so it will look like: `$ fastq_quality_filter –Q33 –q20 –p99 –i inputfile –o A10_qual.fastq` where –Q33 tells it normal sanger sequencing, -q20 means minimum quality of 20, -p99 says at least 99% of bases have to have minimum quality or it will throw out the read. This is very strict.
  - `$ for i in *trim.fastq; do /home/fastx_toolkit-0.0.14/src/fastq_quality_filter -Q33 -q20 –p 99 -i $i -o ${i%trim.fastq}qual.fastq; done` ← this replaces the `trim.fastq` extension with `qual.fastq`
  - the '$i' links the command to the 'i' you gave it in the for loop
  - % → take i and cut off everything after the % and replace it with whatever is after the } (so this adds a new extension to the new file)
  - Look at the number of reads remaining after the filter, and probably should note this
- If you used PEAR and are working with merged reads, you may need to use PRINSEQ (or Kircher scripts) instead of FastX toolkit. Something weird happened to the assembled files… Missy suggests filtering out by exact duplicates as well as 3' and 5' duplicates
- Keep all sequences in the same directory because downstream analyses require them
- Get the reference genome.
- BWA mem on the filtered fastq files (use bwtsw algorithm to index genome; bwa mem is an algorithm with long-read support > 100bp and split alignment; recommended for high-quality queries b/c it's more accurate). This will align and make sam files
- For paired end, reads 1 and 2 must be in same order: "If *mates.fq* file is absent and option **-p** is not set, this command regards input reads are single-end. If *mates.fq* is present, this command assumes the *i*-th read in *reads.fq* and the *i*-th read in *mates.fq* constitute a read pair. If **-p** is used, the command assumes the 2*i*-th and the (2*i*+1)-th read in *reads.fq* constitute a read pair (such input file is said to be interleaved). In this case, *mates.fq* is ignored. In the paired-end mode, the **mem** command will infer the read orientation and the insert size distribution from a batch of reads."
- If you built your own reference (e.g., by concatenating baits with Ns in between them), you will need to index your reference first with `bwa index ref.fasta`

- `$ for i in *qual.fastq; do bwa mem -M -P -a -t 10 path/to/ref/HcZfUnix/HcZfUnix $i > ${i%qual.fastq}sam 2> ${i%qual.fastq}mem.log; done`
- and for paired `$ for i in *R1_trim.fastq; do bwa mem -M -P -a -t 10 /HcZfUnix/HcZfUnix $i ${i/R1_trim/R2_trim} > ${i%.fastq}.sam 2> ${i%.fastq}.mem.log; done`
    - When substituting, be careful with syntax: If you have a sample name that includes the text "R1" anywhere, it will get confused if you only say substitute R1 with R2
    - '`reads.fq mates.fq`' are files specifying the read1 and read2 for each sample and should come after the reference directory
    - *no path to specific file in HcZfUnix; use the whole folder (bwa needs several of the files). So if you are using tab-complete, you'll get the HcZfUnix/ directory, then when you type Hc and tab-complete, it will give you HcZfUnix. and you have to delete the '.'
    - –M: mark shorter split hits as secondary (for Picard compatibility; important for functionality with MarkDuplicates/ identifying PCR duplicates); –t: number of threads (check # processors first. Linux server has 2 processors, 6 cores each, so theoretically we can do 12 threads (but recommended to use 10 or so. Super memory intensive!). –P: In the paired-end mode, perform SW to rescue missing hits only but do not try to find hits that fit a proper pair; –a: Output all found alignments for single-end or unpaired paired-end reads. These alignments will be flagged as secondary alignments.
    - > means "put stuff in here" or "save as"
- Samtools view to make bam files that we can use for GATK and other downstream analyses `$ for i in *.sam; do samtools view -bt /HcZfUnix_ref/HcZfUnix.fasta -o ${i%sam}bam $i; done`
- If you mapped single end and paired end data separately in bwa, merge them now.
    - `$ samtools merge out.bam bam1 bam2`
- Add readgroups and sort bam with Picard Tools
    - `$ for i in *.bam; do java -Xmx2g -jar /home/genetics/picard-tools-1.123/AddOrReplaceReadGroups.jar INPUT=$i OUTPUT=${i%bam}tag.bam SO=coordinate RGID=${i%bam} RGLB=1 RGPL=illumina RGPU=1 RGSM=${i%bam}; done`
    - RGID=Read Group ID. Until now we have just been setting this equal to the individual, but you can do something like modern vs ancient or high/low/mid. This modifies the bam headers so that they are "tagged" with the read group ID (necessary for downstream analyses). It would be useful to set up my sample names so that after every second underscore was the RGID.
    - This step also sorts by where the reads align in the amakihi (or reference) genome.
- Mark PCR duplicates with Picard Tools
    - `$ for i in *.tag.bam; do java -Xmx2g -jar /home/genetics/picard-tools-1.123/MarkDuplicates.jar INPUT=$i OUTPUT=${i%tag.bam}rmdup.bam METRICS_FILE=${i%tag.bam}rmdup.metrics ASSUME_SORTED=true; done`
    - This step removes PCR duplicates. can look at metrics file for statistics on how many were duplicates, etc.
- Index sorted, duplicate-filtered bam with samtools (this creates the .bai file)
    - `$ for i in *.rmdup.bam; do samtools index $i; done`

Instead of GATK, you can use the steps outlined on the Palumbi website (http://sfg.stanford.edu/SNP.html). The objectives are to 1) merge alignment files and realign poorly mapped regions, 2) detect variant sites and filter out true sites from false positives, 3) extract genotype information for all individuals at all variant sites, 4) calculate allele & genotype frequencies, 5) perform Principal Component Analysis to find large-scale differences between populations, and 6) perform FST outlier analysis to find loci potentially under selection.

GATK objectives:
1) realign (-take out indels… this extension has to be called .sort.rmdup.intervals)
2) filter (-remove malformed reads)
3) genotype (-generates .vcf file; then we use vcf tools to see how many SNPs & further subset/analyze)

- GATK RealignerTargetCreator (this step takes a long time). Note: in the ancient merged pipeline, this step produced 95% sequences that failed Mapping Quality Zero filter. Need to figure out how that influences what happens downstream…
    - `$ for i in *.rmdup.bam; do java -jar /home/genetics/GATK/GenomeAnalysisTK.jar -T RealignerTargetCreator -nt 10 -R /mnt/HcZfUnix_reference/HcZfUnix.fasta -I $i -o ${i%rmdup.bam}sort.rmdup.intervals; done`
- GATK IndelRealigner and malformed read filter (this step takes a long time)
    - `$ for i in *rmdup.bam; do java -jar /home/genetics/GATK/GenomeAnalysisTK.jar -T IndelRealigner --filter_bases_not_stored -R /HcZfUnix_ref/HcZfUnix.fasta -I $i -targetIntervals ${i%.rmdup.bam}.sort.rmdup.intervals -o ${i%bam}realign.bam; done`
    - IndelRealigner uses funky parallelization so doesn't accept the –nt flag. `filter_bases` flag tells the process: "if a read has no stored bases (i.e. a '*'), filter the read instead of blowing up".
- GATK UnifiedGenotyper: "The system can be very aggressive in calling variants. We use extensive post-calling filters to eliminate most of the False Positives" (Broad Institute)
    - `$ java -jar /usr/local/bin/GATK/GenomeAnalysisTK.jar -T UnifiedGenotyper -R /mnt/ebs/HcZfUnix_reference/HcZfUnix.fasta -I M1_R1_qual.rmdup.realign.bam -I M2_R1_qual.rmdup.realign.bam -I M5_R1_qual.rmdup.realign.bam -I M6_R1_qual.rmdup.realign.bam -I M22_R1_qual.rmdup.realign.bam -o 111413_amakihi_6k_read1.vcf -filterMBQ -nt 8 --min_base_quality_score 20 -stand_call_conf 20 -stand_emit_conf 20`
    - needs the .bai files to be in the same folder; max # threads UnifiedGenotyper can support is 8.
    - Don't combine R1s and R2s. minimum quality = 20; minimum quality to print = 20. Of course, if you have more than a few realigned files, you don't want to type them all. TC has a script called 'lastpipe' that automates this process for each file in the folder.

```
#!/bin/bash
CUR=pwd
REFERENCE=/home/July14/amakihi/HcZfUnix/HcZfUnix
GATKCOMMAND=/home/GATK/GenomeAnalysisTK.jar
COMMAND="java -Xms18G -Xmx18G -jar ${GATKCOMMAND} -T UnifiedGenotyper -R
${REFERENCE}.fasta -PF gatk_perf_031215.log"
# find all the rmdup.realign.bam's we want
for FILE in 'find . | grep .realign.bam';
do
COMMAND="${COMMAND} -I ${FILE}"
done
COMMAND="${COMMAND} -o amakihiN144_R1R2_SNPs_031215.vcf -filterMBQ -nt 10 --
min_base_quality_score 20 -stand_call_conf 20 -stand_emit_conf 20"
$COMMAND
```

- You should do additional quality filtering and manipulation during/ after these GATK steps.
    - some options are: –sn specifies particular samples, so if you want to just do this to 2 samples you list them individually (-sn SAMPLE_A –sn SAMPLE_B); select any sample that matches an expression and sites where the QD annotation is >10: -se 'SAMPLE.+PARC' –select "QD > 10.0"
- Using the reference (-R), select only SNPs & MNPs that are multi-allelic (i.e., SNPs w/ >1 allele listed in the ALT column), which will get rid of indels automatically, and exclude non-variant loci (-env)
    - `$ java –Xmx2g –jar /home/genetics/GATK/GenomeAnalysisToolkit.jar –R /path/to/ref.fasta –T SelectVariants --variant input.vcf –o output.vcf -selectType SNP -selectType MNP -env`

- GATK defines a SNP as a base that is different from the reference, so you'll get bases that are SNPs in your dataset but also bases that are fixed in your dataset but differ from the reference. Genotypes are 0/1 if homozygous for reference/alternate allele, 1/1 if homozygous for alternate allele, ./. if missing
- Now we can filter to get rid of indels, low quality samples, etc.—using GATK and then vcftools
- GATK Variant Filtration (different from SelectVariants because this keeps & flags the variants not passing filters, & adds annotations in the filter fields)

`$ java -Xmx2g -jar /path/to/GATK/GenomeAnalysisToolkit.jar -R /path/to/ref.fasta -T VariantFiltration --variant input.vcf -o output.vcf --filterExpression "QD < 2.0 || FS > 60.0 || MQ < 40.0 || HaplotypeScore > 13.0 || MQRankSum < -12.5 || ReadPosRankSum < -8.0" --filterName "TECfilters"` where `QUAL`=quality; `DP`=global depth of coverage; `AF`=allele frequency; `AD`=allelic depth; etc. This will throw warnings b/c many of these flags are evaluated only at heterozygotes. To avoid getting a million (literally) warnings that "RPRS does not exist", add the argument `2>/dev/null` to the end in order to redirect warnings to an output file. Others at http://gatkforums.broadinstitute.org/discussion/1268/how-should-i-interpret-vcf-files-produced-by-the-gatk. My two statements were: `(java -Xmx2g -jar /home/GATK/GenomeAnalysisTK.jar -R /home/July14/amakihi/HcZfUnix/HcZfUnix.fasta -T VariantFiltration --variant amakihiR1_SNPs_070615.vcf -o amakihiR1_SNPsfiltered_070815.vcf --filterExpression "QD < 2.0 || FS > 60.0 || MQ < 40.0 || HaplotypeScore > 13.0 || MQRankSum < -12.5 || ReadPosRankSum < -8.0" --filterName "TECfilters" 2>/dev/null &)` followed by `(java -Xmx2g -jar /home/genetics/GATK/GenomeAnalysisTK.jar -R pathto/ref.fasta -T SelectVariants --variant input.vcf -o output.vcf --excludeFiltered &)` The 'excludeFiltered' flag includes TECfilters as well as the QD, FS, MQ, etc.

  - `||` means you are selecting variants that have at least one of the conditions met (the logical "or"); `&&` requires that both conditions are met.
  - The GATK people strongly suggest using the recalibration tool, but I don't think I can with my dataset. If I can't; here are their recommendations for STARTING out but I MUST try different values to see which are best for my data.

Here are some recommended arguments to use w VariantFiltration when ALL other options are unavailable:

Filtering recommendations for SNPs:

- `QD < 2.0` `MQ < 40.0` `FS > 60.0` `MQRankSum < -12.5` `ReadPosRankSum < -8.0`
- For example, suppose I want to select all of the sites where sample NA12878 is homozygous-reference. This can be accomplished by assessing the underlying VariantContext as follows:
- `java -Xmx4g -jar GenomeAnalysisTK.jar -T SelectVariants -R b37/human_g1k_v37.fasta-variant my.vcf -select 'vc.getGenotype("NA12878").isHomRef()'`
- We have put in convenience methods so that one can now filter out hets (`isHet == 1`), refs (`isHomRef == 1`), or homs (`isHomVar == 1`). For hets you can filter over all sample genotypes using something like `'GT == 0/1'`.
- Now here's a more sophisticated example of JEXL expression that finds all novel variants in the total set with allele frequency > 0.25 but not 1, is not filtered, and is non-reference in 01-0263 sample:
- `! vc.getGenotype("01-0263").isHomRef() && (vc.getID()==null || vc.getID().equals(".")) && AF> 0.25 && AF < 1.0 && vc.isNotFiltered() && vc.isSNP() -o 01-0263.high_freq_novels.vcf -sn 01-0263`
- GATK SelectVariants to grab just the variants within homozygous region
  - `$ java -Xmx2g -jar /usr/local/bin/GATK/GenomeAnalysisToolkit.jar -R /path/to/ref.fasta -T SelectVariants --variant input.vcf -o output.vcf -L my.intervals`
  - You first need to create a file my.intervals with one interval per line in this format:
  - -chr1:from-to
  - -chrx:from-to
  - etc.
- vcf-stats will give you general statistics on the run (how many heterozygotes per site, etc)
  - Vcftools uses its own perl, so you might get errors about vcf.pm. The fix: "For running the Perl scripts, the PERL5LIB environment variable must be set to include the Vcf.pm module" so in

.bash_profile, add '`export PERL5LIB=./vcftools_0.1.12b/perl`'. Alternatively, you can simply type this into the command line and then run the vcf-stats or query.

- vcf-tools --vcf file.vcf --out output_prefix [filtering options] [output options]
  - you can subset data (certain chroms or individuals), analyze (Fst, HWE and more), and convert files here
  - Get minimum depth of coverage: First use GATK's CoveredByNSamplesSites tool to generate a file of intervals (a list of SNP sites): `java -Xmx2g -jar /GATK/GenomeAnalysisTK.jar -T CoveredByNSamplesSites -R /amakihi/HcZfUnix/HcZfUnix.fasta -V amakihi40kSNPsR1.vcf -out amakihiN144_R1SNPs_DP8.intervals -minCov 8 –nt 10 --percentageOfSamples 0.001` The percentageOfSamples is the % required to have this minimum coverage at these sites. I prefer to do these 2 steps separately so I set it at a minimum corresponding to 1 individual or fewer. The output will be a list, typically 10% of the original # of SNPs, that looks like this: chr1:186189 etc., with a new site on each line. In order to use this site list in vcftools, you have to convert it to a tab-delimited file: `cat amakihiR1SNPsDP8.intervals | awk -F ":" '{print $1 "\t" $2}' > amakihiR1SNPsDP8_intervals.txt` And now you can use that list for vcftools processing: `$ /vcftools/bin/vcftools --vcf input_file.vcf --positions interval_file --recode --out amakihiN144R1SNPsDP8` or use --bed instead of --positions if you have a bed file.
  - If you have known family groups, it's a good idea to remove the SNPs that do not follow Mendelian inheritance patterns (typically 5 - 10% of SNPs). You can do this with a built-in tool in vcftools. My approach was to create a vcf that was just for the individuals in the family group, find and output loci that violate Mendelian assumptions, and use that locus list to exclude loci from my final vcf with all individuals. I think this requires an older version of vcftools but I can't remember why… `$ vcftools-master/src/cpp/vcftools --vcf amakihiN9_R1R2DP9_80pct_famgroup.recode.vcf --out family_group --mendel amakihi_familygroup_80pct_cut.ped` where .ped is the file giving family relationships.
  - Remove individuals that were sequenced poorly, as they will significantly reduce the number of loci in analyses if kept in the dataset: `$ /path/to/vcftools/bin/vcftools --vcf input_file.vcf --missing-indv --out outfile_prefix`; then make a list of individuals to discard based on your criteria, followed by `$ /path/to/vcftools/bin/vcftools --vcf input_file.vcf --remove indivs_to_remove.txt --recode --out prevfile_50pctind`
  - Filter missing data with vcftools: `$ /path/to/vcftools/bin/vcftools --vcf input_file.vcf --keep keep_indivs.txt --remove discarded_indivs.txt --recode --max-missing 0.8 --out outfile_prefix` (this number is kind of backwards: requires floating point # between 0 and 1, where 1 is no missing allowed)
  - I do several iterative steps here. For instance, first I remove individuals that are missing 75% or more sites; then I remove sites that are genotyped in fewer than 40-50% of individuals. Next I remove individuals missing 45% ore more sites, and finally use only those sites that are genotyped in 80-90% of remaining individuals.
  - It's a good idea to use vcftools' or GATK's vcf validators. `$ java -Xmx2g -jar /home/GATK/GenomeAnalysisTK.jar -R /HcZfUnix/HcZfUnix.fasta -T ValidateVariants --variant amakihiN123R1R2_25pcind80pcloc50pc80pc.recode.vcf --validationTypeToExclude CHR_COUNTS --warnOnErrors` where CHR_COUNTS are the # AC, AN and throw errors if you've recoded with recode-INFO-all. You want it to warn, not abort, each time an error is found.

END OF SNP PROCESSING SECTION; BEGINNING OF SNP ANALYSIS SECTION

- get allele frequency data for one or all chromosomes `$ vcftools --vcf input_file.vcf --freq --out allele_freqs`
- `$ vcf-stats amakihi_file.vcf > amakihi.stats.txt`
- calculate Fst by creating two popfiles, one with each population, and using `/home/vcftools_0.1.12b/bin/vcftools --vcf amakihiR1DP8.recode.vcf --max-missing 0.9 --weir-fst-pop popfile_high.txt --weir-fst-pop popfile_low.txt --out pop1-pop2`

- to find regions of homozygosity in the genome, or "Long Runs Of Homozygosity" (LROH): substitute fst flag for --LROH and needs a specific chromosome to test. Later you can concatenate them all together to find regions consistent across populations. It is helpful to remove inbred individuals first; these are those that have a large number of homozygous regions. You can count the number of occurrences of each indiv in a file with `$ grep -o -c indiv_name filename.txt`
- to get individual heterozygosity, replace --freq with --het. To get a list of SNPs in the same order as BayeScan (also has functional utility as suggested), use `--site-quality` (Generates a file containing the per-site SNP quality, as found in the QUAL column of the VCF file); to get mean coverage depth per indiv `--depth`; to get depth per site summed across individuals `--site-depth`; to get mean depth per site across individuals `--site-mean-depth`; to get nucleotide diversity per site `--site-pi`; to get allele frequency per site `--freq`; to get allele frequency counts (useful for verifying BayeScan) `--counts`; to get the density of SNPs in a certain window `--SNPdensity 100000`; to get a list of singleton SNPs `--singletons`; to extract information about DP or other info in the FORMAT field per site/individual `--extract-FORMAT-info DP` (awesome!)
- If you want mean nucleotide diversity (in the 3rd column of the *.sites.pi files), you can do this quickly with awk: `for i in *80pctloc.sites.pi; do awk '{total += $3} END {print total/NR}' $i > ${i%.sites.pi}.mean.pi; done` ('total' is a variable created by adding each element of column 3; then when that is done print the value resulting from dividing that variable by the number of rows) and then you might want to put these in a file for later reference: `for i in *80pctloc.mean.pi; do cat $i | awk -F '\t' -v x=$i '{print x "\t" $0}' > $i.named; done` and then `cat *80pctloc.mean.pi.named > allpops_80pctloc.mean.pi`
-

- vcf-query to get just the chr, pos and genotype for each sample. GT is GenoType, and the brackets loop over all samples.
  - `$ vcf-query –f '%CHROM:%POS\t%REF[\t%GT]\n' 110613_amakihi_snponly.vcf > 110613_vcf_genotypes.txt`
  - `$ vcf-query –l input.vcf > headerfile.txt` will generate a list of indivs in the order of the vcf file. Add chrom:pos and REF to the beginning of the file in 2 lines, and then transpose to create a tab-delimited header file (`tr '\n' '\t' < infile > outfile`). Concatenate the first vcf-query above (without the –l flag) onto the header (`cat headerfile genotypesfile > genotypes_with_header.txt`).
- bcftools will generate a file of 0s and 1s rather than letters (same command)
- Useful tools here are 1) taking the first line of a file (e.g., if you want to copy the header into a new file): `$ head -n 1 amakihiN144_gtypes_header.txt > headerfile.txt` or 2) deleting everything but the first line in a file (make sure you are saving this to a new file!): `sed –i '1!d' unix_file.txt`
- Count the number of columns so you can figure out the next few steps: `awk '{print NF}' filename.ext | sort -nu | tail -n 1`
- Generate genotype files for each sample separately: `$ for i in {3..23}; do cat 110613_85pcent_genotypes.txt | awk -v x=$i '{print $1 "\t" $x}'> "m"$i"_85pcent.txt"; done` (Awk uses x but bash uses i, so we are telling awk to treat 'i' like 'x'). **NOTE: the m refers to column number. So you will have to make sure you know what column corresponds to what sample, and then re-label them. Alternatively, to re-name them after the 2nd column in the header: `$ for i in *txt; do d="$(head -1 "$i" | awk '{print $2 "_" $3}').txt"; mv "$i" "$d"; done`
- Generate files of all het sites and hom sites for each individual: `$ for i in {3..6}; do cat amakihi_gtypes.txt | awk -F '\t' -v x=$i '{split($x, a, "/"); if (a[1] != a[2]) print $1 "\t" $2 "\t" $x}' > "column"$i"_75pct-shared-loci_hets.txt"; done` –this generates a file that includes only heterozygous sites for each individual. To do this for all individuals and name the files according to the column/header name, use this command with the loren2.awk script below. `for i in {3..146}; do awk -f loren2.awk -v x=$i amakihiN144_gtypes.txt; done` → find this file with `find /directory/ -name 'loren2.awk'`

```
BEGIN{
        FS="\t"
}
NR==1 {
        name=$x;
        print (name) > (name ".homs");
        next
}
{
split($x, a, "/"); if (a[1] == a[2]) print ($1 "\t" $2 "\t" $x) >> (name ".homs")
        }
END {
}
```

- Edit the script so it applies to heterozygotes (change instances of ".homs" to ".hets"; change `a[1]==a[2]` to `a[1]!=a[2]`); run it again. This will produce a het file & a hom file for each individual.
- Now we want to convert these to bed files so we can bin SNPs. `$ for i in *hets.txt; do cat $i | awk -F '\t' '{split($1, a, ":"); print a[1] "\t" a[2] "\t" a[2]+1 "\t" $2 "\t" $3}' > "${i%txt}bed"; done` –this generates a .bed file of het sites for each individual; do the same for homs. This creates bins of 1 bp (a[2]+1); can create bins of any size.
- Use bedtools with 80k_not_baits_unique_80bp.bed to get coverage of baits by samples. Bedtools converts bam to bed (here we figure out how many of our SNPs match up with the baits)
- You can use `sed -i '1d' file.txt` to remove the first line of a file (where -i tells it to modify the file rather than printing to screen), so all together `$ for i in *hets.bed; do sed -i '1d' $i; done`
- `for i in *hets.bed; do coverageBed -a $i -b 100k_bins.bed > ${i%bed}cov; done` –this generates a .cov file (coverage) of heterozygous sites for each individual. 100k_bins.bed is a file w/ 100k bp intervals in which the SNP sites will be counted (the coverage of SNPs in an interval). You will get errors if any of the files is empty (e.g., if an individual has 0 het sites)
    - So you move from your 2 bed files, which have all hets and all homs, to 2 cov (coverage) files in the 100k_bins step above. The file format will be:
      chrom chrom# start bp end bp 0 0 10000 0.00000
      where #hets in each 100kb region
      The position along chromosome is the same in the hom and het files. The positioning part allows you to paste them together and then calculate het/hom ratios.

For the coverage files from coverageBed, we want to add a column with the file/sample name, and then combine hets & homs into the same file (lined up by bin), and concatentate all those files into one input for R

- To generate a new column that has the sample name in each line of the file: `$ for i in *.cov; do cat $i | awk -F '\t' -v x=$i '{print $0 "\t" x}' > $i.name; done` (where print $0 means to print the whole line.)
- We want to generate a file that has a list of het and hom sites at each position (or a count of het/hom sites in each region), so we combine the files by pasting one file to the right of another file. `$ paste file1 file2 > new_file` or, for many samples: `$ for i in *.hets.cov.name; do paste $i ${i/hets.cov/homs.cov} > ${i%hets.cov.name}hetshoms.txt; done`
- Now we want to remove duplicate columns from the file so it has each piece of information only once.
    - The easiest way to do this is with awk print or a simple cut: `cat filename | cut –f2,3,4,9,11` (no spaces!) will print the columns listed and then you can save it as a new file name. For multiple files: `$ for i in *.name; do cat $i | cut -f1,2,3,4,12,16 > "${i%txt}cut.txt"; done` -- Also useful for when you don't know the number of columns and just want to remove a couple is: `$ cut -f1-4,7- file.txt > newfile.txt` (this will put columns 1 through 4 and 7 through the end of the file into a new file)

- To replace filenames/ content within a file: `sed 's/old/new/g' input.txt > output.txt` --e.g., `for i in Honeycreeper*; do cat $i ${i/Honeycreeper/Passerines} > ${i/Honeycreeper/honeypass}; done`

NOTE: `cat FileB >> FileA` adds `FileB` to the end of `FileA`, but `cat FileB > FileA` creates a new file named `FileA`; so be careful not to replace an existing file. To concatenate multiple files at a time, use: `cat file1 file2 file3 > newfile`, which will put them in the order in which they are typed.

- `$ cat *.name > allsamples_baits.cov` To concatenate multiple files in the order you want them in, use: `cat file1 file2 file3 > newfile`, which will put them in the order in which they are typed. This is the file that you want to put into R for the tile plots.
- I have had a difficult time with ggplot and subsetting, so thought it might be easier to add a column onto this file that had the heterozygosity ratio (hets/(hets+homs)). Theoretically this works, but not if you have zeroes in the homs column (and I do): `$ cat amakihi_cov_file | awk -F '\t' '{$(NF+1)=$4/($4+$6);}1' > amakihi_hetratio.txt`
- To convert the vcf file into other formats (e.g., Genepop, BayeScan, Arlequin), use the program PGDSpider2. You have to start by generating a "spid" file for the program to read. The easiest way to do this is use the PGDSpider GUI to generate the file (a basic text file), and then edit it on your own. Then you can convert: `$ java -Xmx96000m -Xms512m -jar ./PGDSpider2-cli.jar -inputfile amakihiN144_SNPsR1DP6.vcf -inputformat VCF -outputfile amakihiN144R1_BS.txt -outputformat GESTE_BAYE_SCAN -spid vcf_to_BS.spid` --This takes a ton of memory. I was not successful in creating a .str file that would run in fastStructure (although it ran in regular Structure), so instead I had to use PLINK to go from vcf to .bed, .fam, .map and .ped which are the alternatives to .str in fastStructure (and also used in dissect and probably other software). Note: PLINK requires numerically named chromosomes, so if there is anything more complicated in your dataset that you want to retain, they need to be renamed. e.g., chr1_rand could be chr101, chr1A could be chr1000 and chr1A_rand could be chr1001, or some similar coding.
- To replace carriage returns with commas (e.g., to edit a spid file): `tr '\n' ', ' < input.txt > output.txt` (but this didn't add a space after the comma)
-
- For HPC nodes, you need a qsub file to start a job. I am not sure the most efficient way to do this, but I have one for Structure, for each K separately. I copied the first one with `$ for i in {3..12}; do cp structure-himem_K2.qsub structure-himem_K$i.qsub; done` Better yet, make the loop within the qsub file in the command. Then submit the job with `qsub jobname.sh`
- ..

POTENTIAL ANALYSES

- **Outlier** detection in BayeScan. After downloading BS, compile it using 'make' (if you get an error about lpthread, you have to first yum-install or wget the glibc-static package). Then you should just be able to `source/bayescan_2.1 amakihi_input_file.txt –o output_prefix` or for multiple files, `for i in *25pctind_R1R2DP9_BS.txt; do source/bayescan_2.1 $i -o ${i%BS.txt}out; done`
  - Unfortunately the locus list becomes dissociated with the Fst values, so you need to go back into the original vcf file to find out the order in which the loci were input. This can be done using one of vcftools' options (e.g., --freq).
  - Then you can extract the outlier locus from this file with `sed -n '10242p' file.frq` (this will print line 10242, i.e., SNP 10241 (there is a header line)) but it would be useful to have only the right elements from all loci at once. TC has a solution, of course! First, add SNP numbers to the .frq file by adding line numbers (-n) minus 1 (awk statement; don't want to number the header): `cat -n x.frq | awk '{$1-=1}1' OFS='\t' > x.frq.num`. Then use awk to find lines in the .frq file where the first field (now the line number) is one greater than the first field in file 1: `awk 'FNR==NR{a[$1]++;next}a[$1]' outlier.file x.frq.num | awk '{print $2 "\t" $3}' >`

`x.outlier.snps`. Translation: For each record or line (`FNR==NR`), loop through the first field in the outlier file (`{a[$1]++;`), then consider the first field in the second file (`next}a[$1]`), and finally print columns 2-3.

- Outlier detection in Lositan: requires a .gen genepop input file (one locus per line, then genotypes). The SNP numbers are dissociated from the actual chrom:pos so it's necessary to have generated a .keptsites file in vcftools. Use the list in Lositan to extract certain sites from the .keptsites file:
  - `sed -n '10226,10242p' file.keptsites` (this will print lines 10226 through 10242, where 10226 is SNP_10226)
- Outlier detection and assessment of genetic structure in TESS3: requires data in R's .geno format; I converted using the LEA package. I had to install this with Bioconductor (not R's install.packages, in which case the package is not found): `source("https://bioconductor.org/biocLite.R") / biocLite("LEA")` --In this package there is a simple tool vcf2geno that will convert, but **memory maxes out at 5000 loci**: TESSfile <- vcf2geno("/path/to/file.vcf"). Next simply run from within the TESS3-master directory `$ ./TESS3 -x amakihi_recoded.geno -r latlong_noindivnames.txt -K 4 -i 100000` where 'i' is the number of iterations (although this never exceeded 100 for me…?). This is very fast and generates the Q matrix and a file of Fst values for each locus.
-
- **PCA** in R (prcomp or dudi.pca in ade4)
  - Need genotypes with numbers, not letters, so use the --012 output option in vcftools. But BEWARE! This codes missing data as -1, which is numerical, so it disproportionately influences where individuals fall out on the PCA.
- **Overrepresentation Test**: assess whether loci invoked in adaptation (selection, differentiation, outliers, etc. from above) represent certain gene ontology/ molecular functions more often than expected by chance (i.e., the number of genes in each category in the reference genome)
  - Blast against the SwissProt database: `for i in *.fa; do ../../../ncbi-blast-2.2.30+/bin/blastx -query $i -out ${i%fa}blastxSPout -db ../Uniprot/uniprot_sprot.fasta -outfmt "6 qseqid sseqid evalue"; done`
  - Get the list of genes from blast results: `for i in *blastxSPout; do cat $i | cut -f2 | awk -F '|' '{print $3}' > ${i%out}.genenamestaxa; done` will give geneID_taxon and then get the gene names only with `for i in *genenamestaxa; do cat $i | awk -F '_' '{print $1}' > ${i%taxa}; done` --I combine genes from multiple populations with `cat *SP.genenames > all_mostnegTajD1kbinblastxSP.genes`, then sort them with `sort -k1,1 all_mostnegTajD1kbinblastxSP.genes > allpops_mostnegTajD1kbinblastxSP.genes.sort` and filter for only unique genes `uniq allpops_mostnegativeTajD_1kbin.blastxSP.genes.sort allpops_mostnegativeTajD_1kbin.blastxSP.genes.unique` (some overrepresentation software will do this, but not all, so I find it best to do it myself).
  - Compare this list of genes with the chicken gene list from the whole genome: I did this in Panther (pantherdb.org) after downloading a list of chicken genes and extracting their names (Gallus_gallus.Gallus_gallus-5.0.genelist). You can perform a statistical overrepresentation test, export the results as a table (but it's difficult to preserve the hierarchical classification), and view the results in chart form. The pie charts are pretty useless, but you can get bar charts for target species and then repeat the analysis with the reference to visualize the categories that are over or underrepresented.
  - Assess whether any gene ontology categories are over (or under) represented. Under-represented is difficult when you have so few genes to start with. I could pool genes from all analyses and that might help increase the power to detect underrepresented ontologies.
- **GWAS** and other analyses in dissect or other software
  - In PLINK, test associations by creating a phenotype file (needs to have fields for Family ID (default in vcf is same as individual ID), Individual ID, phenotype (1=unaffected, 2=affected, 0=missing). You need to give it a flag for a separate phenotype file: `/home/genetics/plink-1.07-x86_64/plink --file amakihiN6R1R2_DP9mendelrm25pctind80pctloci50pctind_expinf`

`--assoc --pheno expinf_phenotypes.txt --out amakihiN6R1R2…_expinf_out` and can do other options like exact tests and a logistic model. ("The `--logistic` command may give slightly different results to the `--assoc` command for disease traits, but this is because a different test/model is being applied (i.e. logistic regression rather than allele counting)"). This is very fast but produces an output file plink.assoc that is multiple-space delineated, which I'm having trouble reading in R. I wanted to do a fisher exact test using the `--fisher` flag (which you can't do in v1.9 without specifying a chromosome set for non-humans). I was getting the error code "Error FEXACT code 3" which was extremely cryptic and difficult to Google. I hypothesized it meant I had monomorphic alleles in the dataset, which I shouldn't after creating a file specifically for those 6 individuals, but… I fixed (presumably) the problem by adding the MAF flag: `./plink --file /Loren/ExpInfDP9mendelrm_ExpInfSNPs80pctloc --fisher --pheno expinf_phenotypes.txt --maf 0.01 --out ExpInfDP9mendelrm_ExpInfSNPs80pctloc` Unfortunately this reduced the dataset by ~10% (5k loci out of 40k)

- You can sort the file by p values or allele frequency to get a list of sites that are fixed differences between groups: `sort -k 3,3 myfile > myfile_sorted` (you need the column number twice to tell it where to stop sorting). If you have a header, `cat file | head -n +2 | sort -k 9,9 > outfile.` **Unfortunately the output files are space-delimited rather than tab-delimited, so you have to process the files before sorting.** I confirm it wasn't sorted properly with `grep missing_pvalue file`. You should be able to replace multiple spaces by tabs with `sed -i .backup 's/ \+ /\t/g' assoctest.txt` (.backup tells sed the extension for backup files), but this isn't working for me.
- I can use 'tr' to replace the spaces with tabs (`tr ' ' \\t < ExpInf_assoc.txt > ExpInf_assocv2.txt`), but now there are multiple tabs in a row and R still can't read it. I used `unexpand -a oldfile > newfile` to replace whitespace with tabs, cut to remove the first (unnecessary) column, and then a python script to remove white space at the beginning of each line (this was causing problems with the next step). Then I used the 'split' tool to generate a .bed file as previously. Looks ok but when I try to sort by OR, it's not correct. :(Unexpand seems to only get rid of the first whitespace instead of all of them, even with the -a ('all') flag.
- 'Squeeze' (the -s flag from translate) is a nice option that gets rid of all but the first whitespace: `tr -s " " < ExpInf.input > Expinf.squeezed`, and then those can be replaced with tabs: `tr " " "\t" < Expinf.squeezed > Expinf.squeezed.txt` and finally sorted. A reverse sort might be a nice way to keep the header on the top. I believe sort numbers fields starting with 0. To generate the bed file: `cat *fixeddiffsites | awk -F '\t' '{split($1, a, ":"); print a[1] "\t" a[2] "\t" a[2]+1}' > ExpInf.assoc.fixeddiffsites.bed`
- After the bed file is generated, you can get the flanking regions of the SNPs to blast them and see what they match. This requires two steps:
- 1) use extract_flanking.sh which contains the following script: `cat yourfile.bed | awk '{var1=$2; var1-=50; var2=$3; var2+=49; print $1 "\t" var1 "\t" var2}' > yourfile_50bp.bed`
- 2) `bedtools getfasta -fi HcZfUnix_reference/HcZfUnix.fasta -bed ExpInf_highOR400bflanks.bed -fo ExpInf_highOR400bflanks.fa`
- then you can blast against a custom or existing database: `/home/ncbi-blast-2.2.30+/bin/blastx -query ExpInf_highOR400bflanks.fa -out ExpInf_highOR400bflanks_blastout.txt -db nr -remote -evalue 1e-3` (`-outfmt 6` puts the output in a nice table but then you have to search for the queries. -outfmt 7 is tabular w/ comment lines. Then 6&7 can be additionally formatted). If no outfmt flag, can add `-line_length 70` to lengthen description. blastx is a translated nucleotide query against a protein database; blastn is a nuc query against a nuc database. `-db` asks for the database name: `nr` is the non-redundant protein sequences from several sources; `nt` is the partially non-redundant nucleotide sequences. List of others here: ftp://ftp.ncbi.nlm.nih.gov/blast/documents/blastdb.html. There is a 'human and mouse immunoglobulin variable region nucleotide (igSeqNt) and protein (igSeqProt) sequences'

- fastStructure: not really that fast; about the same as regular Structure w/ 20G RAM (probably also limited by CPU instead of RAM). I haven't found out if there is a way to parallelize it or not… You can use logistic or simple prior: `$ sudo python structure.py -K 8 --input amakihifile_noprefix --output amakihioutput --prior=logistic` where default input is the .bed format, and you can specify `--format=str` instead, if that's what you have. After running under multiple K scenarios, you need to run a separate script that will determine the best K: `$ sudo python chooseK.py --input=HIamakihiN144R1_DP8sites_logc`
- **Structure**: run 3 replicates and this will produce a bunch of log files as part of the output. The script `extract_str_logL_and_K_generic.py` is a script to extract the K and log likelihood from each such file with the command `python extract_str_logL_and_K_generic.py filename.log`. It's not a pretty solution, but it's a solution! To put the resulting files together so all K values are in one file, simply `cat *.logLL`. It may be possible to run the chooseK.py script from fastStructure without much modification, but I'm not sure.
- others
  -

OPTIONAL ADDITIONAL STEPS

- Compare frequencies between two populations:
  - Generate .frq files for each population separately: `$ vcftools --vcf input.vcf --freq --keep pop1 --out pop1`
  - Concatenate the two files together `paste pop1.frq pop2.frq > bothpops_freqdata.txt` and use only the important columns `cat bothpops_freqdata.txt | cut -f1,2,5,6,11,12 > bothpops.frq`
  - Split the allele:freq columns that are :-delimited: `cat bothpops.frq | awk -F '\t' '{split($3,a, ":"); split($4, b, ":"); split($5, c, ":"); split($6, d, ":"); print $1 "\t" $2 "\t" a[1] "\t" a[2] "\t" b[1] "\t" b[2] "\t" c[1] "\t" c[2] "\t" d[1] "\t" d[2]}' > bothpops_freq.txt`
  - Create an additional column on the right (column 11) that calculates the absolute value of freq difference between allele 1 columns. My syntax for absolute value in awk is not correct, so I'll just sort with -g and I think that will work? `cat bothpops_freq.txt | awk -F '\t' '{print $0 "\t" $4-$8}' > bothpops_freqdiffs.txt`
  - It would work except that sort is doing something really bizarre where it's losing a column and gaining a previously deleted column and I don't even know what value it is sorting on (certainly not the one I thought I told it…column 2, I believe). SO weird. You should be able to `sort -g -k11,11 bothpops_freqdiffs.txt > bothpops_freqdiffs.sorted` but this is definitely not working… super bizarre. The -n flag works but this puts the most-negative values at the top of the file and the largest positive values at the end of the file
- Can count the number of times in a vcf file that the reference allele is "A": `grep -v '#' sample.vcf | cut -f 4 | grep -c "A"`
- In awk, '~' means "starts with" so you can find header lines by ~ '#' or `'$1 ~ "#" print $0'`
- grab only the SNPs that map to our baits (from pre-hets file):
  - go to sam_files alignments
  - get honeycreepr_baits.bed; `$ wc -l bedfile.bed` tells you how many of the baits mapped to the genome (in our 6k SNP case it was a very small number, and many off-target SNPs)
  - `$ sortBed -I honeycreeper.bed > honeycreeper_baits_sorted.bed`
  - GAKT SelectVariants to grap SNPs that are in certain intervals (maybe in one chromosome, or one region we are interested in) → change bed file into an interval list: `awk -F '\t' '{print $1 "\" $2 "\" $3}' sorted.bed > baits.interval_list` –here, $1 is column one, so we are telling it to print columns 1 – 3 separated by tabs and saved into a bait interval list file

- `$ java –Xmx2g –jar /usr/local/bin/GATK/GenomeAnalysisToolkit.jar –R /path/to/ref.fasta –T SelectVariants–variant input.vcf –o output.vcf –L baits.interval_list`
-

- Convert amakihi coordinates to zebrafinch coordinates & vice versa
  - For amakihi→ZF: `cat input.bed | perl MMLO.pl s2_chr6.chr6.mapbases.gz > ZFoutput.bed`
  - and for ZF→amakihi: `cat input.bed | perl MMLO.pl chr6.s2_chr6.mapbases.gz > amakihi_output.bed`
- BWA: index reference, produces a .fai file like a dictionary –MAPPING BAITS TO GENOME
  - first convert sam → bam (picard tools)
    - then convert bam → bed (bedtools) – bamtoBed
    - `Get s3://… honeycreeper_baits.fna`
    - `bwa mem –M –t 4 /path/to/reference_folder baits.fna > baits.sam…` (similar to mapping of reads)
    - `samtools view –bt ref.fasta.fai baits.bam …`
    -
-