WORKFLOW

Download .fastq.gz → gunzip → trim → quality filter → align to genome → call SNPs → data analysis!

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

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. You may 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.

Useful tools and tricks:

- ctrl u saves what you've typed; ctrl y pastes it
- check how much space you are taking up on a drive with `$ du -hs /directory/`
- `$ ls -1 *txt | wc –l` counts the number of text files in the directory.
- find a file with `find /directory/ -name 'filename.ext'`
- the '$i' links the command to the 'i' you gave it in the for loop; % means take i & cut off everything after the % and replace it with whatever is after the } (e.g., add a new extension to the new file)
- Count the number of columns: `awk '{print NF}' filename.ext | sort -nu | tail -n 1`

- Log in: `$ ssh user@destination` and supply the pw when prompted
- Install packages with `wget package_name` or `yum install package_name`
- 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`
- Next you have to do QC to figure out where to trim. Running FastQC will produce an html file showing summary statistics of the quality; you can see at what point in the read the quality drops 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.

- Quality filter & 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 html files locally: `scp xyz@123.45:/fq/*.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` (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; for clarity rename to .trim.fastq: `for i in *.fastqNEW; do mv $i ${i%fastqNEW}trim.fastq; done`
  - More generally, rename is something like `$ 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; do one file at a time to avoid gigantic files `for i in *.fastq; do gzip ${i%fastq}fastq.gz $i; done`

- 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 > ../Junerun/$i; done`
- Quality filter trimmed reads with fastq_quality_filter from the FastX toolkit: `$ for i in *trim.fastq; do /home/fastx_toolkit-0.0.14/src/fastq_quality_filter -Q33 -q20 -p 98 -i $i -o ${i%trim.fastq}qual.fastq; done` ← this replaces the `trim.fastq` extension with `qual.fastq`; –Q33 tells it normal sanger sequencing, -q20 means minimum quality of 20, -p99 says at least 99% of bases have to have minimum quality or it will throw out the read. This is very strict.
  - Look at the number of reads remaining after the filter, and note this
- Get the reference genome and index it if it has not been indexed (if not, *.fasta will be the only file)
  - First run `$ bwa index -a bwtsw genome.fasta` which will create *.bwt, *.pac, *.ann, *.amb and *.sa files. Next, run `$ samtools faidx genome.fasta` which will create *.fai. All of these are necessary for the next step. Later you'll need GATK to create *.dict, or you can do it now with `home/gatk-4.1.2.0/gatk --java-options "-Xmx2G CreateSequenceDictionary -R /sackettl/ref.fasta`
- BWA mem on the filtered fastq files (bwa mem is an algorithm for long reads (> 100bp) and split alignment; recommended for high-quality queries b/c it's more accurate). This will align to genome.
  - For paired end, reads 1 and 2 must be in same order: "In the paired-end mode, the **mem** command will infer the read orientation and the insert size distribution from a batch of reads."
  - for unpaired reads `$ for i in *qual.fastq; do bwa mem -v 3 -M -a -t 10 path/to/ref/RefPrefix $i > ${i%qual.fastq}sam 2> ${i%qual.fastq}mem.log; done`
  - and for paired reads `$ for i in *R1_trim.fastq; do bwa mem -v 3 -M -P -a -t 10 /ref/directory $i ${i/R1trim/R2trim} > ${i%.fastq}.sam 2> ${i%.fastq}.mem.log; done` which produces a single paired-end output file but requires the same number of (a corresponding forward and reverse) reads in each input file. '`reads.fq mates.fq`' are files specifying read1 and read2 for each sample and should come after the reference directory. Parameter flags should be listed in this order (v first).
  - –M: mark shorter split hits as secondary (important for Picard compatibility/ functionality with MarkDuplicates); –t: # threads; –P: In paired-end mode, perform SW to rescue missing hits only but do not try to find hits that fit a proper pair; –a: Output all alignments for single-end or unpaired paired-end reads. These alignments will be flagged as secondary alignments.
- Samtools view to make bam files that we can use for GATK and other downstream analyses `$ for i in *.sam; do samtools view -q 30 -bt /HcZfUnix_ref/ref.fasta -o ${i%sam}bam $i; done`
  - If you mapped paired reads and did not quality filter bases/reads previously, do so now by adding the q flag (before the other flags) and a mapping quality threshold, e.g., `-q 30`
- If you mapped single end and paired end data separately in bwa, merge them now. This makes a bam file that is twice as big, so I suspect it is not merging correctly. I have read that picardtools has a better merge option for paired reads.
  - `$ samtools merge out.bam bam1 bam2 -@4` where '@' denotes # of threads for compression
- Add readgroups and sort bam with Picard Tools (this is slow)
  - `$ for i in *qual.bam; do java -Xmx2g -jar /home/picard-tools-1.123/picard.jar AddOrReplaceReadGroups INPUT=$i OUTPUT=${i%bam}tag.bam MAX_RECORDS_IN_RAM=1000000 TMP_DIR=$PWD/tmp SO=coordinate RGID=${i%bam} RGLB=1 RGPL=illumina RGPU=1 RGSM=${i%bam}; done`
  - RGID=Read Group ID. Currently this is set to the individual, but you could do modern vs ancient or high/low/mid (e.g., in sample names, every second underscore is the RGID). This modifies the bam headers so that they are "tagged" with the read group ID (necessary for downstream analyses). `MAX_RECORDS_IN_RAM` is necessary for large files (e.g., over 60 GB); `TMP_DIR` is necessary if there are limits to the number of files allowed in a directory.
  - This step also sorts by where the reads align in the reference genome.

- For some reason you cannot add picardtools to your path because it is not C-compiled, so you have to set an environment variable. On our HPC "the module file uses a method similar to the one picard documentation gives to set up an alias called runpicard" (which they have done for us). So the new command is `for i in *.bam; do runpicard AddOrReplaceReadGroups INPUT=$i OUTPUT=${i%bam}tag.bam SO=coordinate RGID=${i%bam} RGLB=1 RGPL=illumina RGPU=1 RGSM=${i%bam}; done`
- Many enormous temp files are created in this step, so if you install yourself on a HPC, you need to add a tmp directory in a location with unlimited storage so they won't be saved to the compute node. Before my command, I added `cd $PBS_O_WORKDIR \ mkdir -p tmp` and then in my command before the SO part I added `TMP_DIR=$PWD/tmp`

➢ Mark PCR duplicates with Picard Tools

- `$ for i in *.tag.bam; do java -Xmx16g -jar /picard-tools-1.1/MarkDuplicates.jar INPUT=$i OUTPUT=${i%tag.bam}rmdup.bam MAX_RECORDS_IN_RAM=1000000 METRICS_FILE=${i%tag.bam}rmdup.metrics ASSUME_SORTED=true; done`
- This step removes PCR duplicates. You should look at metrics file for statistics on how many were duplicates, etc. The .tag.bam files are needed in the next step.

➢ Index sorted, duplicate-filtered bam with samtools (this creates the .bai file)

- `$ for i in *.rmdup.bam; do samtools index $i; done`

GATK objectives:
1) realign poorly mapped regions (-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)

➢ Note: GATK on our HPC behaves strangely sometimes. In versions 3.5 & 3.7, if the program doesn't recognize the reference it won't throw a useful error, but the logfile will say `Picked up _JAVA_OPTIONS: -XX:+UseSerialGC` (which is normal & also output when the program runs) & the program won't run. In versions 4, the program will run but not to completion. Also, instead of the typical java -jar GenomeAnalysisTK.jar we use locally, on the HPC we invoke gatk with 'rungatk' (it creates the necessary alias).

➢ GATK's indel realignment tools (RealignerTargetCreator and IndelRealigner) are obsolete and unnecessary with HaplotypeCaller

➢ GATK HaplotypeCaller—this will create your initial vcf file; .bai files need to be in the same folder

- `$ ./gatk HaplotypeCaller -R /path/to/ref /ref.fasta -I RSF01.rmdup.bam -I PSVL11.rmdup.bam -O gupd_rangewide_date.vcf --min_base_quality_score 20`
- If you have many realigned files, you don't want to type them all (I don't remember why you can't use *). A script called 'HaploCaller' will automate the process for each file in a folder:

```
#!/bin/bash

CUR=pwd

REFERENCE=/project/reference/ref_prefix
GATKCOMMAND=/project/sackettl/gatk-4.1.2.0/gatk

COMMAND="${GATKCOMMAND} HaplotypeCaller -R ${REFERENCE}.fasta"

# find all the rmdup.bams we have
for FILE in /project/qualfiltered/unpaireddone/taggedbams/dupsmarked/*_rmdup.bam;
do
COMMAND="${COMMAND} -I ${FILE}"
done

COMMAND="${COMMAND} -O GUPD_10rangewide_051519.vcf --min-base-quality-score 30"

$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
  - `/gatk-4.x/gatk SelectVariants --variant input.vcf -R /path/to/ref.fasta --output output.vcf -select-type SNP -select-type MNP --exclude-non-variants true --set-filtered-gt-to-nocall true`
- 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 low quality samples, etc.—using GATK and then vcftools
- GATK VariantFiltration (SelectVariants removes variants not passing criteria; VariantFiltration keeps & flags the variants not passing filters, & adds annotations in the filter fields):
- quality filter based on global & per-genotype criteria (QUAL=quality, DP=depth of coverage (in INFO field); QD = qual / depth; FS = Strand bias estimated using Fisher Exact Test; MQRankSum = mapping quality rank sum test; AD=allelic depth, DP=genotype depth of coverage (in the FORMAT field); 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. So: `/gatk-4.0.1.2/gatk VariantFiltration --variant AKEK_rawSNPs.vcf --output AKEK_rawSNPsQC.vcf -R ../HcZfUnix_reference/HcZfUnix.fasta --filter-name "ReadPosRankSumFilter" --filter-expression "ReadPosRankSum < -8.0" --filter-name "MQRankSumFilter" --filter-expression "MQRankSum < -12.5" --filter-name "FSFilter" --filter-expression " FS > 60.0" --filter-name "QDFilter" --filter-expression "QD < 2.0" --genotype-filter-name "DP4filter" --genotype-filter-expression "DP < 4" 2>/dev/null`
  - I'm not sure if the --genotype-filter-expression parameter works. It didn't in previous versions, but some things have changed. As of early 2019, there are still lots of genotypes <4, and the mean is also <4
  - --`filterExpression` is for the INFO field (global per locus); `--genotype-filter-expression` is for the FORMAT field (specific genotypes)
  - One MUST try different values to see which are best. Some documentation here: https://software.broadinstitute.org/gatk/documentation/article.php?id=11069.
- and now remove flagged variants `/gatk-4.0.1.2/gatk SelectVariants -R pathto/ref.fasta --variant input.vcf --output output.vcf --set-filtered-gt-to-nocall true`
- Alternatively to DP filters, to replace CoveredByNSamplesSites, use DiagnoseTargets for bam files

Filtering options for SNPs:

- Suppose I want to select all of the sites where sample NA1287 is homozygous-reference. This can be accomplished by assessing the underlying VariantContext as follows: `./gatk SelectVariants -R b37/human_g1k_v37.fasta --variant my.vcf -select 'vc.getGenotype("NA1287").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'`.
- GATK SelectVariants to grab just the variants within a certain region
  - `$ ./gatk –R /path/to/ref ref.fasta 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): `$ vcf-stats amakihi_file.vcf > amakihi.stats.txt` (no need to type full path to vcftools). BUT:
  - 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.
- `vcftools --vcf file.vcf --out output_prefix [filtering options] [output options]`
  - you can subset data (by chrom or individual), analyze (Fst, HWE and more), and convert files here
  - The first thing you may want to do is remove all sites that didn't pass all filters. You can do this with `vcftools --vcf infile.vcf --recode --remove-filtered-all` (or `--remove-filtered-geno-all`) `--out output_prefix`
  - 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. I created a vcf that was just for the individuals in the family group, find and output loci that violated Mendelian assumptions, and used that locus list to exclude loci from my final vcf with all individuals. I think this requires an older version of vcftools? `$ vcftools-master/cpp/vcftools --vcf amakihiN9_famgroup.recode.vcf --out family_group --mendel amakihi_familygroup_80pct_cut.ped` where .ped is the file giving family relationships.
  - Select only the sites matching our baits: `$ /path/to/vcftools/bin/vcftools --vcf input_file.vcf --bed bedfile.bed --out out_prefix --recode` (if you have a list of sites instead of a bed file of baits, use `--positions SNP.sites` instead)
  - Filter for minor allele frequency so that you're not picking up artifacts: `$ /path/to/vcftools/bin/vcftools --vcf input_file.vcf --maf 0.1 --out outfile_prefix`
    - Note: 0.1 is a standard threshold, but this still caused some artifacts in my data (missingness of indivs was related to PC scores even in a PCA with a complete dataset), maybe suggesting that indivs w/ maf=0.2 at lots of loci are incorrectly genotyped
  - Remove individuals that were sequenced poorly, as they will significantly reduce the number of loci in analyses if kept in the dataset: `$ /vcftools/bin/vcftools --vcf input.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 --recode --max-missing 0.8 --out outfile_prefix` (the max-missing 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% or more of the remaining sites, and finally use only those sites that are genotyped in 80-90% of remaining individuals. This seems to maximize the number of individuals and sites in the final file.
  - 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

- BaitsTools (github.com/campanam/BaitsTools) to design probes for hybridization capture
  - `ruby baitstools.rb vcf2baits -i path/to/file.vcf --totalvars 60000 --depth 2 --scale -r /path/to/ref.fa -L 120 --lenbef 60 -o GUPD_rangewide --log --distance 25000 --bed --phred64 --gaps exclude --params --complete --noNs --maxgc 55 --maxmask 10.0 --minqual 20` where `--log` outputs a log file; `--bed` lists absolute

coordinates of baits relative to reference; `--params` outputs a table of bait statistics; `--complete` removes candidate baits shorter than the requested length; `--noNs` excludes baits with Ns; `--maxmmask` excludes baits with % masked bases higher than specified value; `--minqual` excludes baits with any base lower than specified quality; `--scale` scales the max # of variants per contig by that contig's length; `-L` is bait length (recommended to use 120bp for modern DNA and 80bp for ancient DNA); `--lenbef` is # bases before the variant to include in the bait sequence; `--depth` is the number of tiled baits per variant; `--distance` is the bp distance between variants on a contig

- other useful options include `--shuffle` to shuffle the last bait forward to compensate for reaching the end of a contig, `--rc` to output reverse complemented baits; `--mint 15` excludes baits with melting temp below specified value; `--offset` is the base pair offset between tiled baits (default 60)

➢ get allele frequency data for one or all chromosomes `$ vcftools --vcf input_file.vcf --freq --out allele_freqs`

➢ To get individual heterozygosity (actually an inbreeding coefficient in this case), 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`

➢ calculate Fst at each site by creating two popfiles, one with each population, and using `/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 remove NA values for sites that are missing data in one pop, create a new file with no NAs: `for F in *.weir.fst; do sed -i.bak '/nan/d' $F; done`

➢ To calculate Hardy-Weinberg equilibrium at each site, use vcftools first: `for i in *recode.vcf; do vcftools --vcf $i --hardy --recode --out ${i%.vcf.recode.vcf}; done`. Then you can filter out only the sites that have a significant heterozygote deficit (in this case, column 7) or all sites out of HWE: `for i in *SNPs.hwe; do cat $i | awk '{ if ($7 < 0.01) print $0}' > ${i%hwe}sig.hwe; done`

➢ To find regions of homozygosity in the genome, or "<u>L</u>ong <u>R</u>uns <u>O</u>f <u>H</u>omozygosity" (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`

➢ 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. This also works on .vcf.gz files.

- `$ vcf-query -f '%CHROM:%POS\t%REF[\t%GT]\n' amakihi_snponly.vcf > vcf_genotypes.txt`
- `$ vcf-query -l input.vcf > indivs-list.txt` will generate a list of indivs in the order of the vcf file. Add chrom:pos and REF to the beginning of the indiv-list 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`
- 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. To re-name them after the 2nd column in the header: $ `for i in *_DP9.txt; do d="$(head -1 "$i" | awk '{print $2 "_" $3}').txt"; mv "$i" "$d"; done`
- Generate files of all het sites and hom sites for each individual and name the files according to the column/header name, use this command with the loren2.awk script below. `for i in {3..146}; do awk -f loren2.awk -v x=$i amakihiN144_gtypes.txt; done`

```
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.
- However, this is going to treat all missing genotypes as homozygous because of the ./. syntax, so you have to create a new file with just the real homozygotes: `for i in *.trim..homs; do grep -v "\./\." $i > ${i%.homs}nomissing.homs; done`
- 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) (and on Mac OSX you need '' after the -i), so all together `$ for i in *hets.bed; do sed -i '' -e '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#      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 concatenate 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 $i | cut -f1,2,3,4,6,7,9,10 > "${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.
- You may want to add a column onto this file that had the heterozygosity ratio (hets/(hets+homs)). This will not work 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)

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. TEC solution: 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 **OutFLANK**: R-based. Will generate input file for you if you have a set of input files. These are basically equivalent to the .012 output from vcftools () plus the associated .indv (make into popNames) and .pos (make into locusNames) files. I created the .SNPmat file by converting vcfs to .012 files and then `sed 's/-1/9/g' input > file.SNPmat`, and then you need to remove the first column of indiv #s, so I did `for i in *.SNPmat; do cut -f2- $i > ${i%SNPmat}newSNPmat; done`. To create the popNames file, I just replaced individual names with pop names. To create the locusNames file, I used several steps. First, use the output file that puts chr# and pos in different columns, and paste them together with a colon using `sed $'s/\t/:/g' infile > outfile`. Next, you need to get them all on one line for reading into R: `for i in *.locusNames; do tr '\n' ', ' < $i > ${i%locusNames}.locusNames; done` which will replace the newlines with commas. Now I think for R to read them, they need quotes around the names, so `for i in *..locusNames; do sed $'s/,/", "/g' $i > ${i%locusNames}.locusNames; done`. Then add one half set of quotes to the beginning of each file, and delete the comma on the very end of the line. Repeat for pop names. Argh; R's 'scan' function separates by commas, so this isn't necessary. It probably would have recognized end lines too...

➢ **Outlier detection** and assessment of **genetic structure** in **TESS3r**, but this requires longitude/latitude coordinates. Memory may max out at 5000 loci (it did in the old conversion step) or some other threshold. This is very fast and generates the Q matrix and a file of Fst values for each locus. The command line version is no longer supported, so I did this all in R (which is also not supported currently). The easiest way is to use .012 files as input, but it will save a ton of time to add individual names in column 1. I did this by creating separate files with the individual names and pop names and then pasting them into a single file.

➢ Assessment of **genetic structure** in **admixture**: This is really easy once you get the file format right – it says it will accept .ped and .geno, but neither seems to work, so I input my .ped into plink and converted it to .bed. Then, if you already know K, you can simply use `$ ../admixture-1.3.0/admixture akek.bed 3` or if you don't, `$ ../for i in {1..10}; do ...`

➢ Assessment of **genetic structure** in **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`

➢ Assessment of **genetic structure** in **Structure**: run 3 replicates and this will produce a bunch of log files as part of the output. The program needs input files mainparams and extraparams as well as a .str file. The .str file should not have a header, but the spid converter gives it a header, so you can remove it with `tail -n +2 original.str > original_nohead.str` . The script `extract_str_logL_and_K_generic.py` is a script to extract the K and log likelihood from each output file with the command `python extract_str_logL_and_K_generic.py filename_f`. To put the resulting files together so all K values are in one file, simply `cat *.logLL`.

- Assessment of **genetic structure** with **PCA** in R (prcomp or dudi.pca in ade4 – dudi.pca always seems to lead to a couple weird outliers)
  - 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)

- Convert files in plink from .ped to .bed with `--make-bed` and include a list of chromosomes so it won't err. Use `--chr-set 54 no-mt no-xy` where the number refers to the autosomal chromosome count and extra flags allow you to specify additional characteristics of your chromosome set. So mine is `../plink_mac/plink --file AKEK_95pctloc --chr-set 54 no-xy --out AKEK_95pctloc`
- 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 <u>tr</u>anslate) 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 script: `cat file.bed | awk '{var1=$2; var1-=50; var2=$3; var2+=49; print $1 "\t" var1 "\t" var2}' > file_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: `/ncbi-blast-2.2.30+/bin/blastx -query ExpInf_400bflanks.fa -out ExpInf_400bflanks_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: <u>ftp://ftp.ncbi.nlm.nih.gov/blast/documents/blastdb.html</u>.

➢

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`

➢ 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'`

➢ 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`

➢