Apr 01, 2019

Working

Genotyping chip data lift-over to reference genome build GRCh38/hg38

Version 2

Kalle Pärn[1], Javier Nunez Fontarnau[1], Marita A. Isokallio[1], Timo Sipilä[1], Elina Kilpelainen[1], Aarno Palotie[2], Samuli Ripatti[2], Priit Palta[2]

[1]FIMM, University of Helsinki, [2]equal contribution; FIMM, University of Helsinki

dx.doi.org/10.17504/protocols.io.xbhfij6

FIMM HumGen Sequencing Informatics

Priit Palta
FIMM, University of Helsinki

ABSTRACT

PROTOCOL STATUS

Working

SAFETY WARNINGS

Requirements and preparatory steps

1 **The protocol is aimed for lifting genotyping chip data from older reference genome build versions over to human reference genome build version GRCh38/hg38.**

The protocol is aimed for lifting PLINK format genotyping chip data from older reference genome build versions over to **GRCh38/hg38** and to VCF format.

For a 'quick and dirty' lift-over you can jump straight to **Step 2** and only run that.
For high-quality and verified results we strongly recommend running at least through **Steps 1-4** and **Steps 9-10**, and if required, run also **Steps 5-8**.
If your PLINK format files have issues (e.g. non-standard allele notations or missing genders), see **Steps 11-13** for troubleshooting.

Throughout the protocol we assume Bash shell.

This **Step 1** defines the requirements for the protocol (e.g. required software packages and reference files) and suggests example commands how to process the files into suitable formats.

1.0 Docker images
We have prepared docker images to ease up the preparatory steps depending on your application:

- genotype-liftover-imputation-protocols-light
- genotype-liftover-imputation-protocols-full

Light image contains the software packages installed (Step 1.1), the reference genome (Step 1.2.6) and example scripts to run commands in each step (Steps 2-9).
Full image contains the software packages installed (Step 1.1), preprepared publically available reference files (as defined in Step 1.2) and example scripts to run commands in each step (Steps 2-9).

Additional to this protocol, the images contain software packages and files required for genotype imputation protocol version 2: https://www.protocols.io/view/genotype-imputation-workflow-v3-0-xbgfijw

The docker images are available at Docker hub.
To pull the image, use the command below and replace <tag> with 'light' or 'full' according to which image you want:

```
docker pull seqinfoteam/genotype-liftover-imputation-protocols:<tag>
```

!! If you don't want to use Docker, you can install the software packages by yourself (Step 1.1), and either download preprepared files (below) or prepare the files by yourself (Step 1.2 and 1.3).

Preprepared reference data (included into the docker images as listed above) are also available for separate downloading at:
https://console.cloud.google.com/storage/browser/fimm-public-data/Imputation/

And example scripts how to run each step at:
https://console.cloud.google.com/storage/browser/fimm-public-data/Imputation/dockers/genotype-chip-data-liftover-protocol/

1.1 Software packages

1.1.1 Download and install the software packages

Required software packages are listed below with the versions that were used in the protocol below. However, use of the newest versions is recommended.

- PLINK v1.9 http://www.cog-genomics.org/plink/1.9/
- PLINK v2.0 (Jan 02 2019 or later version) https://www.cog-genomics.org/plink/2.0/
- R v3.4.1 (or later version) https://www.r-project.org/
- R package data.table https://github.com/Rdatatable/data.table/wiki/Installation
- BCFtools v1.7 (or later version) http://www.htslib.org/download/

1.1.3 Export the paths

Once installed, export the correct paths to environment variable PATH.

```
echo PATH=$PATH:/path/to/plink/:/path/to/R/:/path/to/bcftools/ \
    >> $HOME/.bashrc
source $HOME/.bashrc
```

BCFtools plugin usage requires environment variable BCFTOOLS_PLUGINS exported, e.g:

```
echo export BCFTOOLS_PLUGINS=/path/to/bcftools/plugins >> $HOME/.bashrc
source $HOME/.bashrc
```

1.1.4 Install the R package

Once R is installed, the 'data.table' package can be installed **in R**, e.g.:

```
install.packages('data.table', type = 'source', repos = 'http://Rdatatable.github.io/data.table')
```

1.2. Reference data

Reference variant allele frequency file is required for comparison of the chip genotyped variant allele frequencies in order to identify possible allele swaps and variants with unexpected allele frequency discrepancies.

Note: Chromosome notation in the reference data should follow the GRCh38/hg38 notations ('chr#' for autosomal chromosomes and 'chrX' for chromosome 23).

1.2.1 Obtain the reference variant allele frequency data

If population-specific data are available (e.g. from a corresponding WGS effort) using these data would be preferrable.
Process the data as instructed in **Steps 1.2.2-1.2.4**.

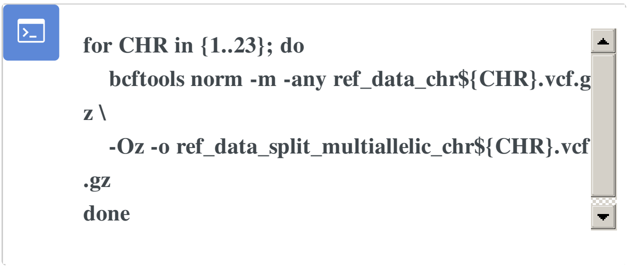
If population-specific reference data is not available, for instance 1000 Genomes Project (www.nature.com/articles/nature15393) data can be used instead.

We have prepared (as described below in **Steps 1.2.2-1.2.4**) the 1000 Genomes Project GRCh38/hg38 data (downloaded from the EBI FTP site: ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/supporting/GRCh38_positions/) to generate variant allele frequency files for 1000GP ALL and EUR samples.

The corresponding allele frequency files are available for downloading at our Google Cloud bucket:
https://console.cloud.google.com/storage/browser/fimm-public-data/1000GP/
If you use these files, you can skip creating the frequency file and **jump to Step 1.3**. Otherwise (if you want to create your own/alternative 1000GP-based custom reference frequency file), follow the steps below.

1.2.2 Check for multiallelic sites

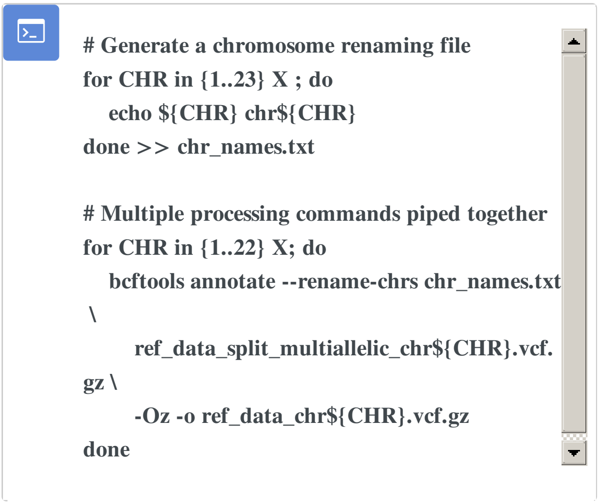
Confirm that multiallelic sites (if present) in your reference data files are decomposed. If they are not, use the example command below to split the multiallelic sites into biallelic records:



```
for CHR in {1..23}; do
   bcftools norm -m -any ref_data_chr${CHR}.vcf.gz \
   -Oz -o ref_data_split_multiallelic_chr${CHR}.vcf.gz
done
```

1.2.3 Check the chromosome notation

Confirm that the chromosome notation in your reference data files follows the GRCh38/h38 notations as **'chr#'** for autosomal, **'chrX'** for chromosome 23 and **'chrM'** for mitochondrial sites. If not, use for instance bcftools to rename the chromosomes:

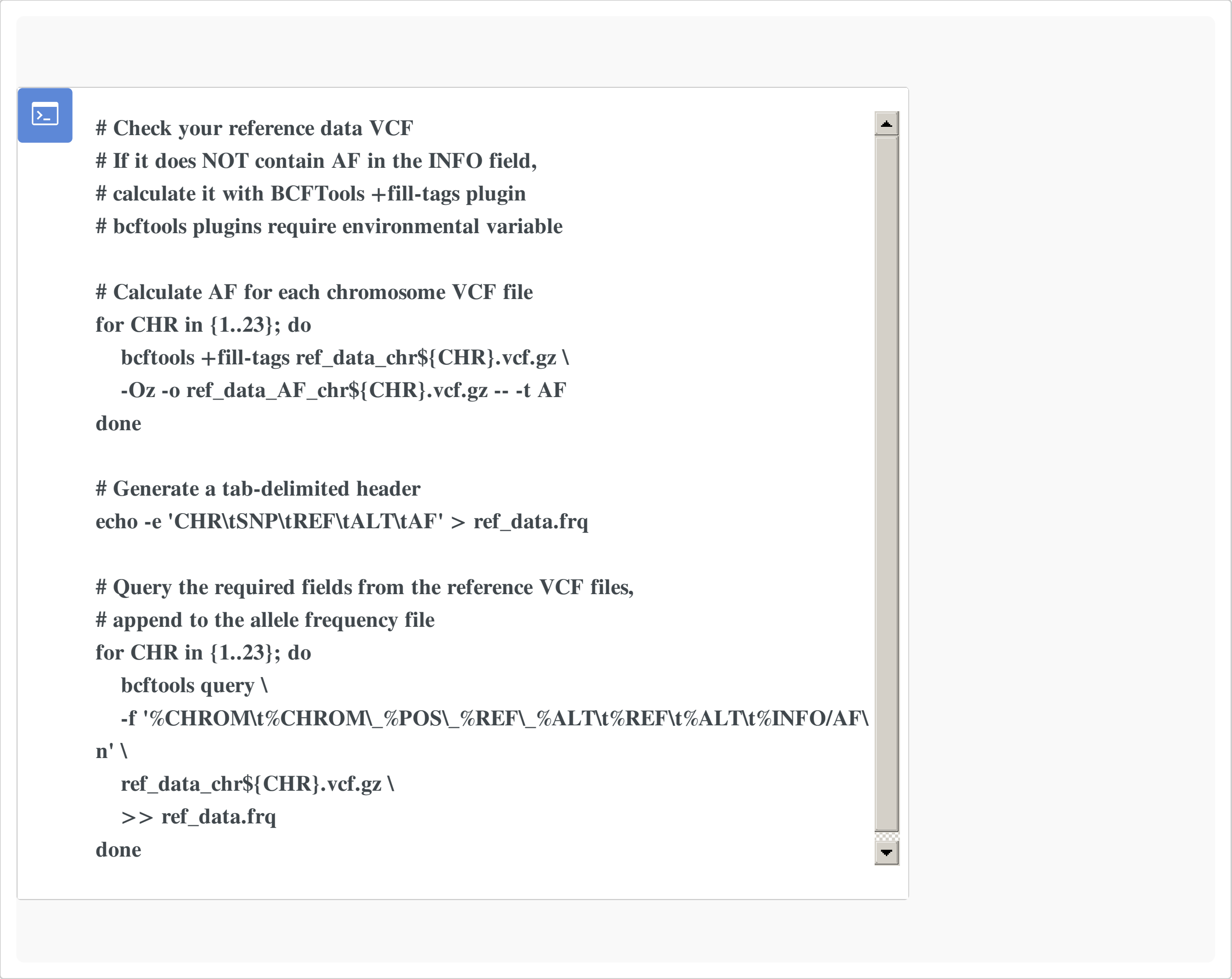


```
# Generate a chromosome renaming file
for CHR in {1..23} X ; do
   echo ${CHR} chr${CHR}
done >> chr_names.txt

# Multiple processing commands piped together
for CHR in {1..22} X; do
   bcftools annotate --rename-chrs chr_names.txt \
      ref_data_split_multiallelic_chr${CHR}.vcf.gz \
      -Oz -o ref_data_chr${CHR}.vcf.gz
done
```

1.2.4 Generate the allele frequency file

Generate a tab-delimited file of the reference data allele frequencies, one line per variant, with columns CHR, SNP (as CHR_POS_REF_ALT), REF, ALT, AF (including the header line).

Use the reference data VCF files as input with the example command below and save the generated frequency file as 'ref_data.frq'.

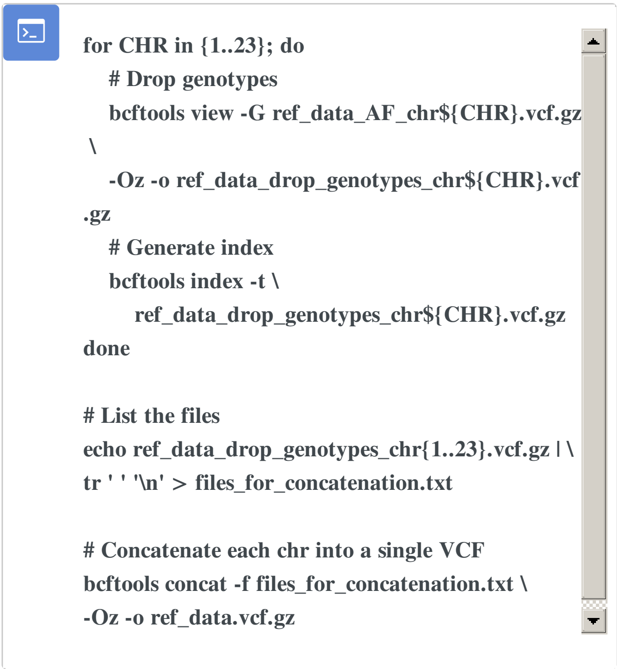


```
# Check your reference data VCF
# If it does NOT contain AF in the INFO field,
# calculate it with BCFTools +fill-tags plugin
# bcftools plugins require environmental variable

# Calculate AF for each chromosome VCF file
for CHR in {1..23}; do
   bcftools +fill-tags ref_data_chr${CHR}.vcf.gz \
   -Oz -o ref_data_AF_chr${CHR}.vcf.gz -- -t AF
done

# Generate a tab-delimited header
echo -e 'CHR\tSNP\tREF\tALT\tAF' > ref_data.frq

# Query the required fields from the reference VCF files,
# append to the allele frequency file
for CHR in {1..23}; do
   bcftools query \
   -f '%CHROM\t%CHROM\_%POS\_%REF\_%ALT\t%REF\t%ALT\t%INFO/AF\
n' \
   ref_data_chr${CHR}.vcf.gz \
   >> ref_data.frq
done
```

Note: Chromosome notation should follow the GRCh38/hg38 notations ('chr#' for autosomal chromosomes and 'chrX' for chromosome 23).

1.2.5 Generate a VCF file without genotypes

In an optional step, a reference VCF file containing RSIDs is required. To operate with smaller files, drop genotypes from your reference data for instance as indicated in the command below:



```
for CHR in {1..23}; do
   # Drop genotypes
   bcftools view -G ref_data_AF_chr${CHR}.vcf.gz
 \
   -Oz -o ref_data_drop_genotypes_chr${CHR}.vcf
.gz
   # Generate index
   bcftools index -t \
      ref_data_drop_genotypes_chr${CHR}.vcf.gz
done

# List the files
echo ref_data_drop_genotypes_chr{1..23}.vcf.gz | \
tr ' ' '\n' > files_for_concatenation.txt

# Concatenate each chr into a single VCF
bcftools concat -f files_for_concatenation.txt \
-Oz -o ref_data.vcf.gz
```

1.2.6 Reference genome fasta

Homo Sapiens assembly hg38 version 0 is used here (match the reference genome always according to your reference data) and the required files are:

- Homo_sapiens_assembly38.fasta
- Homo_sapiens_assembly38.fasta.fai

The files are available for downloading at Broad Insitute storage in Google cloud at https://console.cloud.google.com/storage/browser/broad-references/hg38/v0/?pli=1

1.3. You are ready to start!
As the last prepatory step, let's go over the required input data file(s) and also expected final output files!

1.3.1 Protocol input files

Input data is the chip genotype data in PLINK format (here v1.9).
More details on the PLINK formats can be found at PLINK web page: http://www.cog-genomics.org/plink/1.9/formats

- **<dataset>.bed** - binary representation of genotype calls

- **<dataset>.bim** - extended variant information file including six columns:

1. Chromosome code (either an integer, or 'X'/'Y'/'XY'/'MT'; '0' indicates unknown) or name
2. Variant identifier
3. Position in morgans or centimorgans (safe to use dummy value of '0')
4. Chromosomal base-pair coordinate
5. Allele 1 (corresponding to clear bits in .bed; usually minor)
6. Allele 2 (corresponding to set bits in .bed; usually major)

- **<dataset>.fam** - sample information file including six columns:

1. Family ID ('FID')
2. Within-family ID ('IID'; cannot be '0')
3. Within-family ID of father ('0' if father isn't in dataset)
4. Within-family ID of mother ('0' if mother isn't in dataset)
5. Sex code ('1' = male, '2' = female, '0' = unknown)
6. Phenotype value ('1' = control, '2' = case, '-9'/'0'/non-numeric = missing data if case/control)

1.3.2 Final protocol output files:

The final file here is a VCF format file lifted over to human genome reference build 38 (GRCh38/hg38).

Genome build lift-over

2 If the genotyping chip used an older reference genome version (e.g. GRCh37/hg19) and is in PLINK format, the data have to be lifted over to human genome build version 38 (GRCh38/hg38).

For the genome build lift-over we suggest Will Rayner's method.
Download your genotyping chip specific build 38 zip file (strand and position files) and 'update_build.sh' script from: http://www.well.ox.ac.uk/~wrayner/strand/

To download the files:

```
wget http://www.well.ox.ac.uk/~wrayner/strand/update_build.sh
wget http://www.well.ox.ac.uk/~wrayner/strand/your_chip_platform_specific_chip_strandfile-b38-strand.zip
```

If the genotyping chip platform information is lost, you can try Chipendium to identify the platform based on your data. See Will Rayner's web page for more information: http://mccarthy.well.ox.ac.uk/chipendium/ui/

IMPORTANT NOTES:
- 'update_build.sh' uses hardcoded command 'plink', make sure you have exported the path correctly or replace the plink commands with full path to your PLINK v1.9 installation.
- The method assumes the chip data in Illumina TOP strand format. If this is not the case, you might need to use ILMN or SOURCE strand files, as instructed on the Will Rayner's web page, or follow the optional steps to fix potential issues.

Uncompress the downloaded files and use the .strand file to run the script with the command below.

Input files:
- DATASET = input file prefix of PLINK format files (.bed, .bim, .fam)
- <chip_strandfiles>.zip strand files corresponding the genotyping chip
- <chip_strandfiles>.strand corresponding the unzipped genotyping chip .strand file
- OUTPUT = output file prefix of PLINK format files (.bed, .bim, .fam)

Output files:
- OUTPUT.bed
- OUTPUT.bim
- OUTPUT.fam

```
DATASET=your_dataset_prefix
OUTPUT=your_dataset_output_prefix

# Uncompress the files
unzip <chip_strandfiles>.zip

# Run the script
./update_build.sh \
   ${DATASET} \
   <chip_strandfile>.strand \
   ${OUTPUT}
```

Lift-over verification

3 To confirm the successful genome build lift-over, chip data allele frequencies are compared against the reference data allele frequencies. In this step, chip data allele frequency file is generated.

We define the VCF-style frequency file format for GRCh38/hg38 as follows:

1. CHR - Chromosome code (with 'chr' tag, e.g. 'chr1')
2. SNP - Variant identifier in format CHR_POS_REF_ALT
3. REF - Reference allele (A, C, G, T)
4. ALT - Alternative allele (A, C, G, T)
5. AF - Allele frequency (values from 0 to 1)

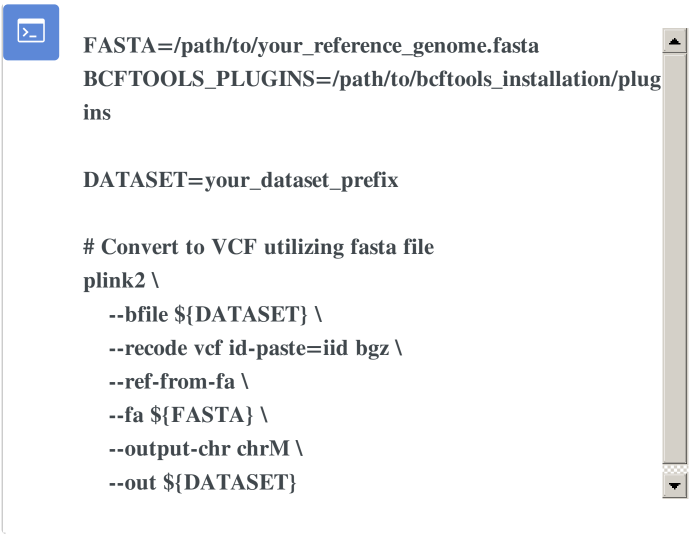
Convert the chip data files to VCF with Plink v2 using the reference genome fasta file to force the correct reference alleles. Calculate the AF values and generate the frequency file.

Input files (from Step 2):
- DATASET.bed
- DATASET.bim
- DATASET.fam

Output files:
- DATASET.vcf.gz
- DATASET_AF.vcf.gz
- DATASET.frq

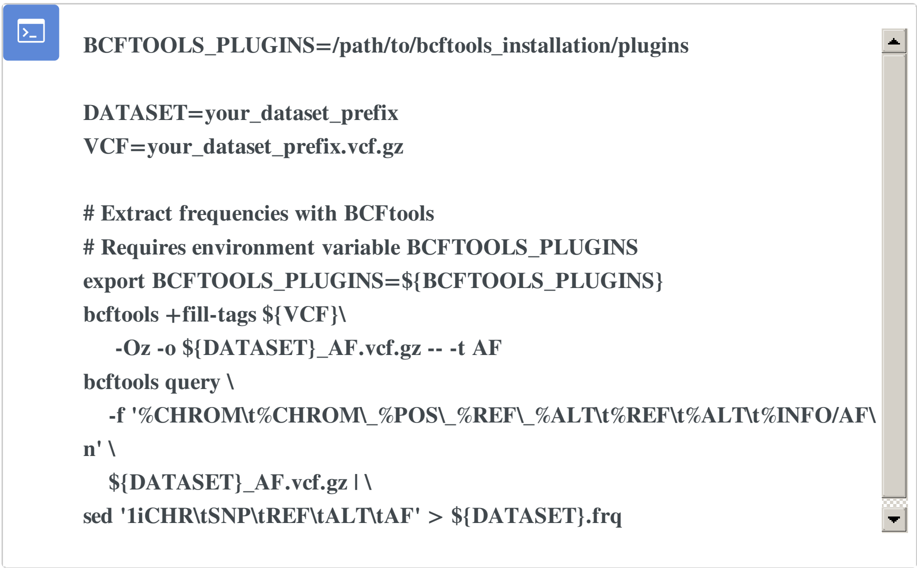
For the most accurate solution, use Plink v2 and BCFtools as follows:

Step 3.1



```
FASTA=/path/to/your_reference_genome.fasta
BCFTOOLS_PLUGINS=/path/to/bcftools_installation/plugins

DATASET=your_dataset_prefix

# Convert to VCF utilizing fasta file
plink2 \
   --bfile ${DATASET} \
   --recode vcf id-paste=iid bgz \
   --ref-from-fa \
   --fa ${FASTA} \
   --output-chr chrM \
   --out ${DATASET}
```

Step 3.2



```
BCFTOOLS_PLUGINS=/path/to/bcftools_installation/plugins

DATASET=your_dataset_prefix
VCF=your_dataset_prefix.vcf.gz

# Extract frequencies with BCFtools
# Requires environment variable BCFTOOLS_PLUGINS
export BCFTOOLS_PLUGINS=${BCFTOOLS_PLUGINS}
bcftools +fill-tags ${VCF}\
    -Oz -o ${DATASET}_AF.vcf.gz -- -t AF
bcftools query \
   -f '%CHROM\t%CHROM\_%POS\_%REF\_%ALT\t%REF\t%ALT\t%INFO/AF\
n' \
   ${DATASET}_AF.vcf.gz | \
sed '1iCHR\tSNP\tREF\tALT\tAF' > ${DATASET}.frq
```

QUICK'n'DIRTY ALTERNATIVE WAY TO GENERATE THE FREQUENCY FILE
Quick and dirty way is to use Plink v1.9 and awk to generate the correct format AF file as indicated in the protocol **Version 1 Step 3** (it is accessible by pressing the Version 2 button on the top left corner of this protocol).
Note: In that approach, the number of variants is duplicated and should be taken into consideration when evaluating the results with the plotting script in the next step!

4 Compare the chip data allele frequencies (created in **Step 3**) to the reference data allele frequencies (created in **Step 1.2.4**).

Copy and save the R script below as 'compare_AF.R' and run it as suggested in the command below.
The R script requires data.table package installed (for suggested instructions, see **Step 1.1**).

Inside the R script, intersection of the reference data and chip data variants (same SNP ID in format CHR_POS_REF_ALT) are formed and corresponding allele frequencies are plotted against each other.

Here, we use 0.1 as a threshold for AF comparison i.e. if chip data AF differs more than 10 pp from the reference data AF, it is marked as discordant. However, less conservative values may be more suitable if a population specific reference data is not available.

Input files:
- DATASET.frq (**Step 3**)
- ref_data.frq (**Step 1.2.4**)

Output file:
- DATASET_AF.png
- *_comparison.txt a log file containing variant counts from the comparison

Inspect the plot:
See example plots at the section 'Expected results' below.

Successful genome build lift-over:
- **Nearly all** chip data variant allele frequencies correlate with the panel variant allele frequencies and data shows tight, uniform diagonal line with increasing slope 1.
-> Continue with **Step 9**

Unsuccessful genome build lift-over:
- **Some** of the chip data variant allele frequencies correlate with the reference data variant allele frequencies, whereas some of the chip data show negative correlation with the reference data variant allele frequencies. Hence, an x-shaped plot is observed. These are likely incorrectly flipped variant alleles and/or ambiguous variant alleles with strand issues.
- **Large proportion** of the chip data variants do not intersect with the reference data variants.
-> Continue with **Step 5** to reflip the problematic alleles.

!! If needed, fix unsuccesful flipping as indicated in the following steps (Steps 5-8).

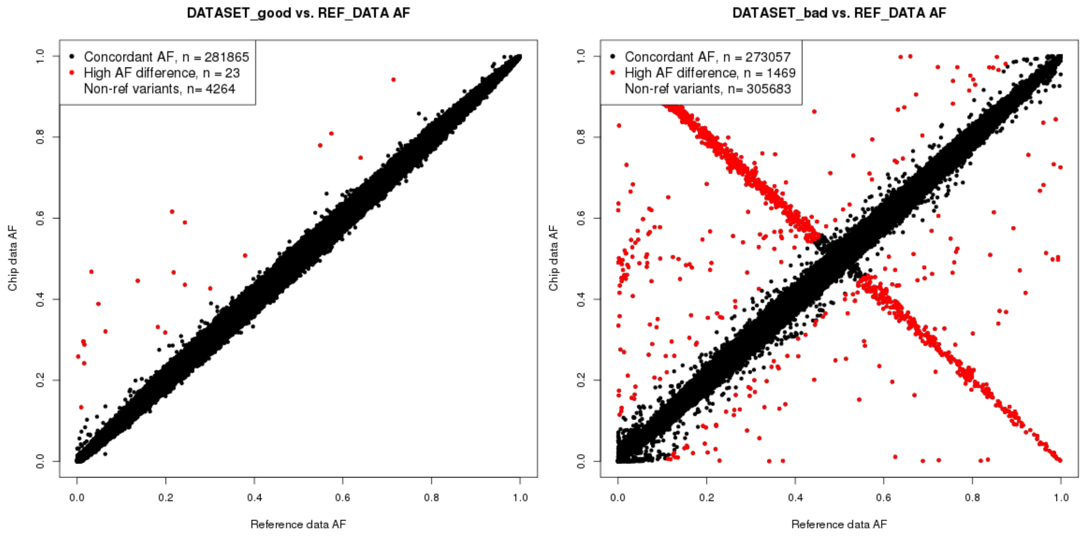
Start by saving the R-script below as 'compare_AF.R.

```
#!/bin/env Rscript --no-save

# Required packages
library(data.table) # For fast fread()

# Input variables
args <- commandArgs(TRUE)
indataset <- args[1]
infile <- args[2]
ref_dataset <- args[3]
refname <- args[4]
af_diff_limit <- as.numeric(args[5])

# Read in the frequency files
chip <- fread(infile, header = T)
ref_data <- fread(ref_dataset, header = T)

# Take an intersection of the reference and chip data
# based on SNP column (in format CHR_POS_REF_AL
T)
isec <- merge(ref_data, chip, by = "SNP")

# Exclude if AF is not within the range
exclude <- !abs(isec$AF.x - isec$AF.y) < af_diff_limit

# Non-ref data variants
nonref <- chip[!(chip$SNP) %in% (isec$SNP)]

# Save the plot as jpg
png(paste0(indataset, "_", refname, "_AF.png"),
   width = 600, height = 600)
# Plot first all and then excludable variants
plot(isec$AF.x, isec$AF.y, col=1, pch=20,
   main=paste0(indataset, " vs. ", refname, " AF"),
   xlab="Reference data AF",
   ylab="Chip data AF")
points(isec[exclude]$AF.x, isec[exclude]$AF.y,
   col=2, pch=20)
# Draw a legend
legend("topleft", legend=c(
   paste0("Concordant AF, n = ", nrow(isec[!exclude])),
   paste0("High AF difference, n = ", nrow(isec[exclude
])),
   paste0("Non-ref variants, n= ", nrow(nonref))),
   col=c("black", "red", "white"), pch=20, cex=1.2)
dev.off()

# Store the high AF difference variants
output <- rbind(c("chip variants", nrow(chip)),
             c("intersection", nrow(isec)),
             c("intersection/chip variants",
                 (nrow(isec))/nrow(chip)),
             c("non-ref variants",
                  nrow(chip) - nrow(isec)),
             c("high AF difference",
                  nrow(isec[exclude])))

      write.table(output,
        paste0(indataset, "_", refname, "_comparison.txt"),
        quote=F, row.names=F, col.names=F, sep = "\t")
```

Save the script as 'compare_AF.R'

Run the above R Chip data compared to p script as follows:

```
DATASET=your_dataset_prefix

Rscript --no-save /path/to/compare_AF.
R \
  ${DATASET} \
  ${DATASET}.frq \
  /path/to/ref_data.frq \
  ref_data_name \
  0.1
```



Successful (left) and unsuccessful (right) genome build lift-over. Chip data compared to population-specific reference data.

OPTIONAL: identify and fix flipping for unambiguous alleles

5 Align the VCF to the reference genome and identify multiallelic sites. Multiallelic sites are formed, when the reference allele at a position does NOT match with the reference genome base.

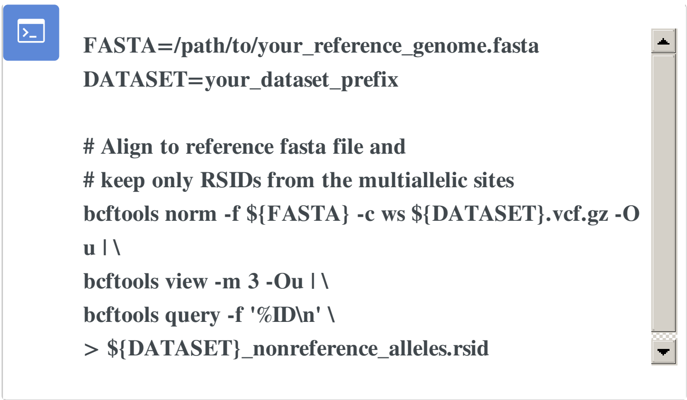
Presumably due to wrong strand orientation in comparison to the lift-over strand file, those alleles were mistakenly flipped in **Step 2**. It is worthwhile to try to flip these variants again to restore the correct strand orientation.

Input files:
- DATASET.vcf.gz (**Step 3**)
- DATASET.bed (**Step 2**)
- DATASET.bim (**Step 2**)
- DATASET.fam (**Step 2**)

- ref_data.frq (**Step 1.2.4**)

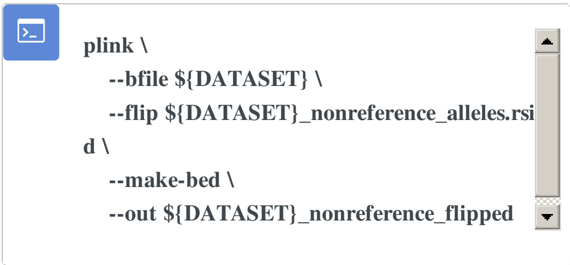
Output files:
- DATASET_nonreference_alleles.rsid
- DATASET_nonreference_flipped.bed
- DATASET_nonreference_flipped.bim
- DATASET_nonreference_flipped.fam

First, identify the sites with a wrong REF allele:



```
FASTA=/path/to/your_reference_genome.fasta
DATASET=your_dataset_prefix

# Align to reference fasta file and
# keep only RSIDs from the multiallelic sites
bcftools norm -f ${FASTA} -c ws ${DATASET}.vcf.gz -O
u | \
bcftools view -m 3 -Ou | \
bcftools query -f '%ID\n' \
> ${DATASET}_nonreference_alleles.rsid
```

Then, flip the identified sites with plink v1.9 (input files generated at **Step 2**):



```
plink \
   --bfile ${DATASET} \
   --flip ${DATASET}_nonreference_alleles.rsi
d \
   --make-bed \
   --out ${DATASET}_nonreference_flipped
```

Finally, confirm the results again as indicated in **Step3** and **Step4**.

go to step #3

go to step #4

OPTIONAL: identify and fix flipping for ambiguous alleles (PLINK format)

6 If an x-shaped plot was observed in **Step 4** (or still after **Step 5**), likely some ambiguous alleles have been wrongly flipped. These may be identified by comparing the AF values to reference data AF and observing inverse values (e.g. most of those marked in red for the bad dataset in the expected results plot in **Step 4**).

Extract ambiguous alleles with inverse AF from the frequency file (generated in **Step 3**) and flip them with plink v1.9.

Input files:
- DATASET.frq (**Step 3**)
- ref_data.frq (**Step 1.2.4**)
- DATASET.vcf.gz (**Step 3**)
- DATASET.bed (**Step 2 / Step 5**)
- DATASET.bim (**Step 2 / Step 5**)
- DATASET.fam (**Step 2 / Step 5**)

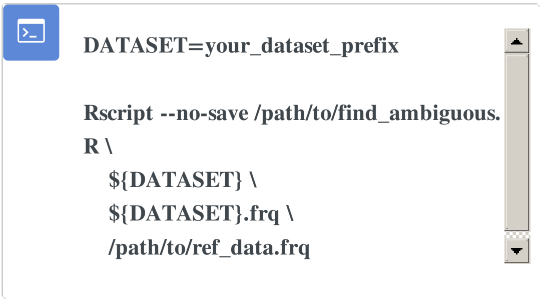
Output files:
- DATASET_flippable_AF.txt
- DATASET_ambiguous_for_flipping.rsid

- DATASET_ambiguous_flipped.bed
- DATASET_ambiguous_flipped.bim
- DATASET_ambiguous_flipped.fam

Start by saving the R-script below as 'find_ambiguous.R':

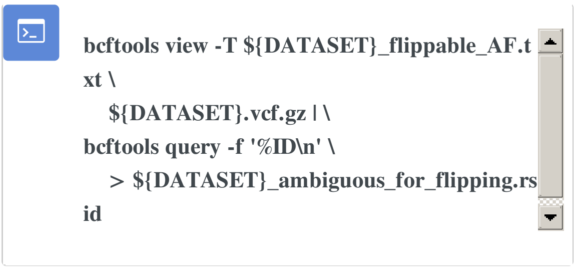
```
#!/bin/env Rscript --no-save

# Required packages
library(data.table)

# Input variables
args <- commandArgs(TRUE)
indataset <- args[1]
infile <- args[2]
ref_dataset <- args[3]

# Read in the frequency files
chip <- fread(infile, header = T)
ref_data <- fread(ref_dataset, header =T)

# Take an intersection of the reference and chip data
# based on SNP column (in format CHR_POS_REF_ALT)
isec <- merge(ref_data, chip, by = "SNP")

# Exclude if AF is not within the range
exclude <- !abs(isec$AF.x - isec$AF.y) < 0.1
discrepant <- isec[exclude]

# Add column with a flipped AF value for the chip data
discrepant$AF_flip <- 1-discrepant$AF.y

# Test if the flipped AF matches to the reference data
flippable <- abs(discrepant$AF.x - discrepant$AF_flip)
   < 0.1

# Keep only ambiguous alleles
ambiguous <- (discrepant$REF.y %in% c("A", "T")
            & discrepant$ALT.y %in% c("A", "T")) |
        (discrepant$REF.y %in% c("C", "G")
            & discrepant$ALT.y %in% c("C", "G"))

# Generate output in format CHR \t POS
output <- strsplit(discrepant[flippable & ambiguous]$SNP, "
_")
output <- do.call(rbind, output)[,1:2]

# Store the output
write.table(output, paste0(indataset, "_flippable_AF.txt"),
quote = F, row.names = F, col.names = F, sep = "\t")
```

Run the 'find_ambiguous.R' as follows:



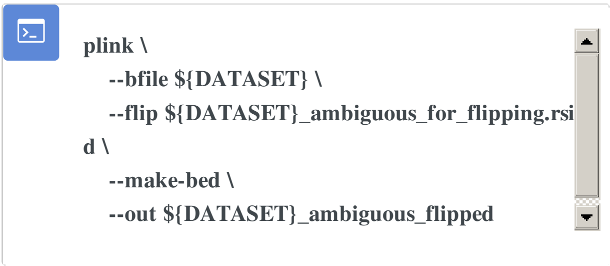
```
DATASET=your_dataset_prefix

Rscript --no-save /path/to/find_ambiguous.R \
   ${DATASET} \
   ${DATASET}.frq \
   /path/to/ref_data.frq
```

Extract the RSIDs from the VCF file (generated in **Step 3**):



```
bcftools view -T ${DATASET}_flippable_AF.txt \
   ${DATASET}.vcf.gz | \
bcftools query -f '%ID\n' \
   > ${DATASET}_ambiguous_for_flipping.rsid
```

Flip the ambiguous alleles in the lifted files (generated in **Step 2** or **Step 5**) with plink v1.9:



```
plink \
   --bfile ${DATASET} \
   --flip ${DATASET}_ambiguous_for_flipping.rsid \
   --make-bed \
   --out ${DATASET}_ambiguous_flipped
```

Confirm the results again as indicated in **Step3** and **Step4**.

go to step #3

go to step #4

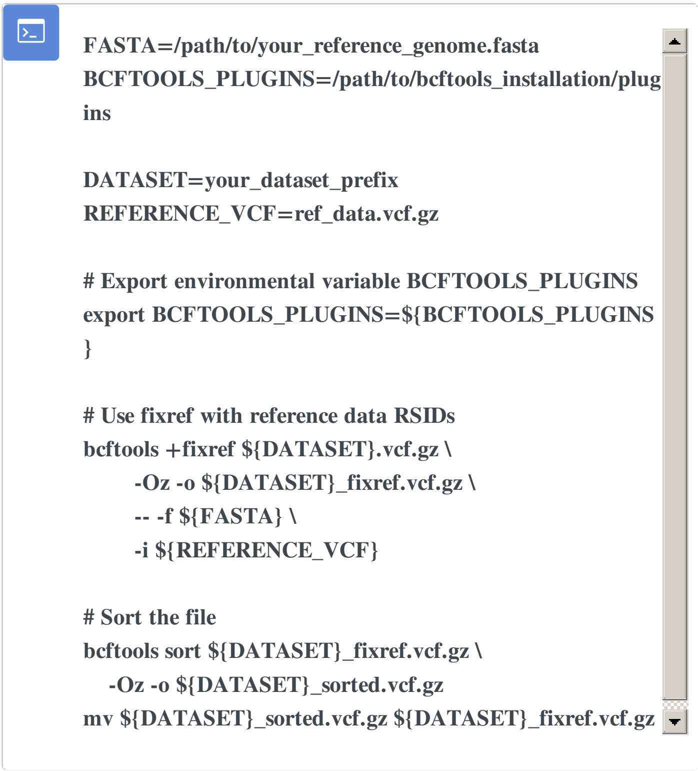
OPTIONAL: allele fixes based on reference data RSIDs (VCF format)

7 More variants can be rescued by comparing the chip data sites to a reference data based on RSIDs. Here, it is important to note, that both datasets use the same RSID versions.

Input files:
• ref_data.vcf.gz (**Step 1.2.5**)
• DATASET.vcf.gz (**Step 3**)

Output files:
• DATASET_fixref.vcf.gz

Use the reference data RSIDs to correct chip data positions and alleles:



```
FASTA=/path/to/your_reference_genome.fasta
BCFTOOLS_PLUGINS=/path/to/bcftools_installation/plugins

DATASET=your_dataset_prefix
REFERENCE_VCF=ref_data.vcf.gz

# Export environmental variable BCFTOOLS_PLUGINS
export BCFTOOLS_PLUGINS=${BCFTOOLS_PLUGINS}

# Use fixref with reference data RSIDs
bcftools +fixref ${DATASET}.vcf.gz \
        -Oz -o ${DATASET}_fixref.vcf.gz \
        -- -f ${FASTA} \
        -i ${REFERENCE_VCF}

# Sort the file
bcftools sort ${DATASET}_fixref.vcf.gz \
    -Oz -o ${DATASET}_sorted.vcf.gz
mv ${DATASET}_sorted.vcf.gz ${DATASET}_fixref.vcf.gz
```

Confirm the results again as indicated in **Step3.2** and **Step4**.

go to step #3

go to step #4

8 Continue to fix the alleles with BCFtools fixref plugin by forcing allele flipping (use with caution and confirm the correctness of the results carefully!).

Input files:
- DATASET.vcf.gz (**Step 7**)

Output files:
- DATASET_fixref_force.vcf.gz

Use BCFtools fixref plugin to force flipping as follows:

```
FASTA=/path/to/your_reference_genome.fasta
BCFTOOLS_PLUGINS=/path/to/bcftools_installation/plugins

DATASET=your_dataset_prefix

# Export environmental variable BCFTOOLS_PLUGINS
export BCFTOOLS_PLUGINS=${BCFTOOLS_PLUGINS}

# Fixref force flip
bcftools +fixref ${DATASET}.vcf.gz \
        -Oz -o ${DATASET}_fixref_force.vcf.gz \
        -- -f ${FASTA} \
        -m flip
```

Confirm the results again as indicated in **Step3.2** and **Step4**.

go to step #3

go to step #4

VCF format and reference genome alignment

9 For any downstream processing, it is advisable to **always** confirm that your VCF data is aligned to the correct reference genome.

As a final step, align the VCF to the reference genome and exclude variants which show wrong reference allele. The alignment also swaps alleles and left-aligns indels where required. To our experience, alignment sometimes forms duplicate records from indels, and thus, duplicate removal is added as a final step.

If you have stopped the lift-over procedure in PLINK format steps and the downstream processes require VCF format, first convert the PLINK format files to VCF format as indicated in **Step 3**.

Input files:
- DATASET.vcf.gz (your final step taken)

Output files:
- DATASET_aligned.vcf.gz

Align to the reference genome with BCFtools, keep only bi-allelic sites (i.e. remove those where REF does not match the reference genome) and remove any duplicate records (considering CHR, POS, REF, ALT to preserve multiallelic sites if present):

```
FASTA=/path/to/your_reference_genome.fasta

DATASET=your_dataset_prefix

# Align to reference genome
bcftools norm -f ${FASTA} -c ws ${DATASET}.vcf.gz -Ou | \
bcftools view -m 2 -M 2 -Ou | \
bcftools norm -d none -Oz -o ${DATASET}_aligned.vcf.gz
```

Confirm the results again as indicated in **Step3** (ignore the Plink command and run only BCFtools command) and **Step4**.

go to step #3

go to step #4

Lift-over confirmation

10 Once you have successfully lifted over your chip data, verify the lift-over by checking variant positions and alleles.
For instance, compare the positions and alleles in your lifted data to known GRCh38/hg38 positions and alleles e.g. from dbSNP https://www.ncbi.nlm.nih.gov/SNP/.

OPTIONAL: data consistency verification (PLINK format)

11 **Steps 11-13** are meant to tackle possible issues we have observed in various legacy data sets in PLINK format.

If for some reason your **.fam** file does not include the sex information (e.g. a VCF converted to PLINK format files loses the sex information), this information need to be added into the **.fam** file from a separate file.

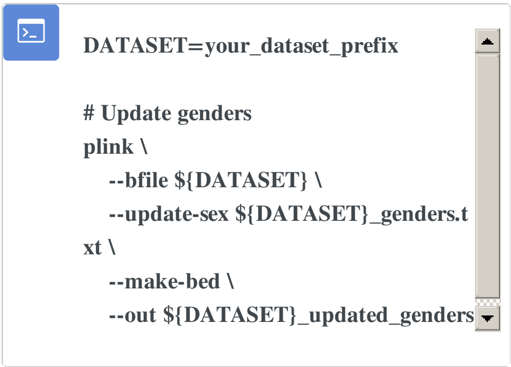
Confirm that the sex information is present in the 5th column for all samples.
If not, obtain the sex information from the original source of your data and update the **.fam** file as suggested in the comman below.

Input files:

- DATASET_genders.txt - containing FID and IID in the first two columns and sex information in the third column as 1 = male, 2 = female or 0 = ambiguous
- DATASET.bed
- DATASET.bim
- DATASET.fam

Output files:

- DATASET_updated_genders.bed
- DATASET_updated_genders.bim
- DATASET_updated_genders.fam



```
DATASET=your_dataset_prefix

# Update genders
plink \
    --bfile ${DATASET} \
    --update-sex ${DATASET}_genders.t
xt \
    --make-bed \
    --out ${DATASET}_updated_genders
```

12 **Steps 11-13** are meant to tackle possible issues we have observed in various legacy data sets in PLINK format.

Sometimes the alleles are marked for instance as '**B**', and such variants should be excluded.

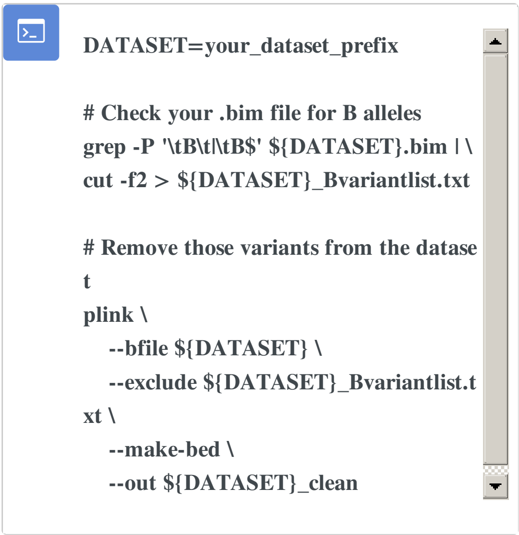
Confirm that the alleles are in standard format i.e. **A**, **C**, **G** or **T**.
If not, correct the inconsistency by removing variants with non-standard allele notation as suggested in the command below.

Input files:

- DATASET.bed
- DATASET.bim
- DATASET.fam

Output files:

- DATASET_Bvariantlist.txt
- DATASET_clean.bed
- DATASET_clean.bim
- DATASET_clean.fam



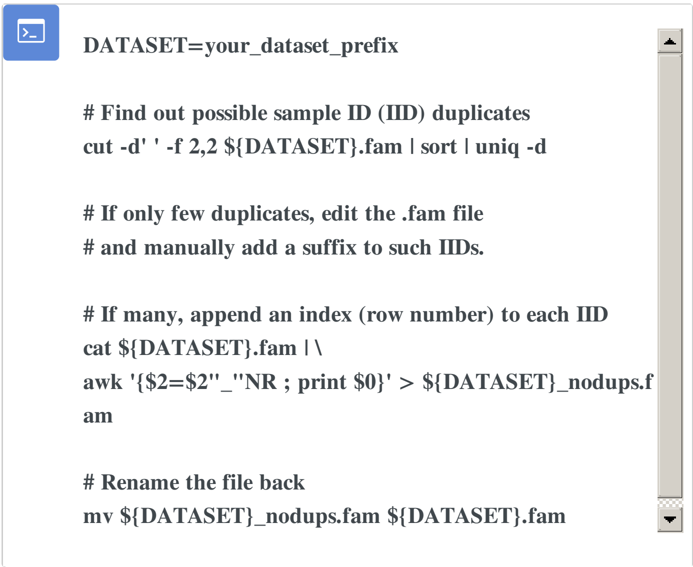
```
DATASET=your_dataset_prefix

# Check your .bim file for B alleles
grep -P '\tB\t|\tB$' ${DATASET}.bim | \
cut -f2 > ${DATASET}_Bvariantlist.txt

# Remove those variants from the dataset
plink \
    --bfile ${DATASET} \
    --exclude ${DATASET}_Bvariantlist.txt \
    --make-bed \
    --out ${DATASET}_clean
```

13 **Steps 11-13** are meant to tackle possible issues we have observed in various legacy data sets in PLINK format.

Make sure no duplicate samples exist. Redundant individual IDs (IIDs) can cause some issues when working with VCF files. In case of duplicates, consider excluding them or adding a running number to those sample/individual IDs in the **.fam** file.

Input file:
• DATASET.fam

Output file:
• DATASET_nodups.fam



```
DATASET=your_dataset_prefix

# Find out possible sample ID (IID) duplicates
cut -d' ' -f 2,2 ${DATASET}.fam | sort | uniq -d

# If only few duplicates, edit the .fam file
# and manually add a suffix to such IIDs.

# If many, append an index (row number) to each IID
cat ${DATASET}.fam | \
awk '{$2=$2"_"NR ; print $0}' > ${DATASET}_nodups.fam

# Rename the file back
mv ${DATASET}_nodups.fam ${DATASET}.fam
```