Variant calling

There are several strategies for variant calling.

Samtools/BCFTools SNP and INDEL calling

Workflows from the htslib

```
#!/bin/bash -l
#SBATCH -p short -N 1 -n 1 -c 4 --mem 16gb --time 2:00:00
#SBATCH -J bcftools_call
#SBATCH -o logs/%x.%j.log
module load samtools
module load bcftools

set -euo pipefail
CPU=${SLURM_CPUS_PER_TASK:-1}
GENOME=S_enterica_CT18.fasta

# need to make a string which is all the bam files you want to process
# but if we do *.bam it will catch the intermediate bam files that are in the folder
m=""
for a in $(cat acc.txt)
do
  m="$a.bam $m"
done

VCF=Salmonella.vcf.gz
VCFFILTER=Salmonella.filtered.vcf.gz
bcftools mpileup -Ou -f $GENOME $m | bcftools call --threads $CPU -vmO z -o $VCF
tabix -p vcf $VCF
bcftools stats -F $GENOME -s - $VCF > $VCF.stats
mkdir -p plots
# plot-vcfstats needs python3 with matplotlib; don't stop the job if it fails
plot-vcfstats -p plots/ $VCF.stats || echo "plot-vcfstats failed - skipping the plots"
bcftools filter -O z -o $VCFFILTER -s LOWQUAL -i'%QUAL>10' $VCF
```

Advanced - GATK variant calling

An existing framework that works can be checked out from the class examples repository https://github.com/biodataprog/GEN220_2026_examples - see the `Variants` folder (the `bcftools` script above is in `Variants/bcftools/`).

Make sure you are running this in ~/bigdata or somewhere with enough space as this will generate large files. The job scripts write their logs to `logs/`, so run `mkdir -p logs` in the `Variants` folder before submitting anything. See UNIX IV for how to choose `-c`, `--mem` and `--time`, and for how job arrays work. The pipeline scripts shown below test for unset variables and missing files themselves (e.g. `if [ -z "$N" ]`), so unlike the template in UNIX IV they don't use `set -euo pipefail`; if you write your own pipeline, start from the template.

The first script `pipeline_GATK/00_index.sh` will download the genome, index and download the fastq files from NCBI SRA. If you had different datasets you would develop your own data files and script.

You don't need to copy this code - do the git checkout and then you can run these steps:

```
cd ~/bigdata/gen220
git clone https://github.com/biodataprog/GEN220_2026_examples.git   # or: cd GEN220_2026_examples; git pu
cd GEN220_2026_examples/Variants
mkdir -p logs
```

This has a configuration file which defines some variables used by the pipeline.

```
GENOMEFOLDER=genome
REFGENOME=genome/Af293_ASM265v1.fasta
GENOMENAME=Af293
SAMPFILE=samples.csv
FASTQFOLDER=input
FASTQEXT=fastq.gz
UNMAPPED=unmapped
UNMAPPEDASM=unmapped_asm
ASMEXT=fasta
ALNFOLDER=aln
ALNTOOL=bwa
HTCFOLDER=cram
HTCEXT=cram
HTCFORMAT=cram
TEMP=cram
GVCFFOLDER=Variants
VARIANTFOLDER=Variants
TREEDIR=strain_tree
RGCENTER=NCBI
RGPLATFORM=Illumina
GVCF_INTERVAL=1
FINALVCF=vcf
PREFIX=Afum_v1
REFNAME=AF293-REF
SLICEVCF=vcf_slice
SNPEFFOUT=snpEff
snpEffConfig=snpEff.config
SNPEFFGENOME=AfumigatusAf293_NCBI
GFFGENOME=Af293_ASM265v1.gff
```

You can customize that as you need for your own data.

The `samples.csv` has a header and is 4 samples for strains from the fungus *Aspergillus fumigatus*.

```
Strain,Filebase
CEA10,SRR7418934
ISSF_21,SRR4002443
1F1SW_F4,SRR4002444
IFM_60237,DRR022927
```

Step 1. `pipeline_GATK/00_index.sh`

Run this as `sbatch pipeline_GATK/00_index.sh` - you will need to wait for it to finish before doing step 2. This step uses the 4 strains which are already on the cluster. Or if you are on your own computer and have the sratoolkit (fastq-dump) installed it will download that.

```
#!/bin/bash -l
#SBATCH -p short -N 1 -n 1 -c 1 --mem 4G --time 2:00:00
#SBATCH -J index
#SBATCH -o logs/%x.%j.log
module load samtools
module load bwa
if [ -f config.txt ]; then
    source config.txt
fi
mkdir -p $FASTQFOLDER $GENOMEFOLDER
pushd $GENOMEFOLDER
```

```
# Download the A. fumigatus Af293 genome and annotation from NCBI RefSeq
# (the same Af293 assembly FungiDB used; FungiDB download links no longer work)
ACC=GCF_000002655.1_ASM265v1
URL=https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/002/655/$ACC
FASTAFILE=$(basename $REFGENOME)
GFF=$GFFGENOME
echo "working off $FASTAFILE - check these match REFGENOME and GFFGENOME in config.txt"

if [ ! -f $FASTAFILE ] ; then
    curl -L $URL/${ACC}_genomic.fna.gz | gunzip -c > $FASTAFILE
fi
if [ ! -f $GFF ]; then
    curl -L $URL/${ACC}_genomic.gff.gz | gunzip -c > $GFF
fi

if [[ ! -f $FASTAFILE.fai || $FASTAFILE -nt $FASTAFILE.fai ]]; then
    samtools faidx $FASTAFILE
fi
if [[ ! -f $FASTAFILE.bwt || $FASTAFILE -nt $FASTAFILE.bwt ]]; then
    bwa index $FASTAFILE
fi

DICT=$(basename $FASTAFILE .fasta)".dict"

if [[ ! -f $DICT || $FASTAFILE -nt $DICT ]]; then
    rm -f $DICT
    samtools dict $FASTAFILE > $DICT
    ln -s $DICT $FASTAFILE.dict
fi

popd
module load sratoolkit
IFS=,
tail -n +2 $SAMPFILE | while read STRAIN SRA
do
    echo $STRAIN $SRA
    # if On UCR HPCC can use already downloaded files
    if [ ! -f $FASTQFOLDER/${SRA}_1.$FASTQEXT ]; then
        if [ -f /bigdata/gen220/shared/data/Afum/${SRA}_1.$FASTQEXT ]; then
            ln -s /bigdata/gen220/shared/data/Afum/${SRA}_[12].$FASTQEXT $FASTQFOLDER
        else
            fastq-dump -O $FASTQFOLDER --split-e --gzip $SRA
        fi
    fi
done
```

Step 2. `pipeline_GATK/01_align.sh`

Run this as

```
# one array task per strain: the 4 lines of samples.csv after the header
sbatch --array=1-4 pipeline_GATK/01_align.sh
```

You will need to wait for it to finish before doing step 3. This step uses the 4 strains. The code at the bottom is for generating a dataset of unaligned reads for further assembly alone.

```
#!/bin/bash -l
#SBATCH -p epyc -N 1 -n 1 -c 16 --mem 32gb --time 8:00:00
```

```
#SBATCH -J bwa
#SBATCH -o logs/%x.%A_%a.log
module load bwa
module load samtools
module load picard
module load gatk/4    # the commands below use GATK4 syntax
module load java

MEM=32g

TMPOUTDIR=tmp

if [ -f config.txt ]; then
  source config.txt
fi
if [ -z $REFGENOME ]; then
  echo "NEED A REFGENOME - set in config.txt and make sure 00_index.sh is run"
  exit
fi

if [ ! -f $REFGENOME.dict ]; then
  echo "NEED a $REFGENOME.dict - make sure 00_index.sh is run"
fi
mkdir -p $TMPOUTDIR $ALNFOLDER $TEMP $UNMAPPED

CPU=${SLURM_CPUS_PER_TASK:-1}
N=${SLURM_ARRAY_TASK_ID:-$1}
if [ -z "$N" ]; then
  echo "cannot run without a number provided either cmdline or --array in sbatch"
  exit
fi

MAX=$(wc -l $SAMPFILE | awk '{print $1}')
if [ $N -gt $MAX ]; then
  echo "$N is too big, only $MAX lines in $SAMPFILE"
  exit
fi

IFS=,
tail -n +2 $SAMPFILE | sed -n ${N}p | while read STRAIN FILEBASE
do

  # BEGIN THIS PART IS PROBABLY PROJECT SPECIFIC
  # THIS COULD NEED TO BE CHANGED TO R1 R2 or R1_001 and R2_001 etc
  PAIR1=$FASTQFOLDER/${FILEBASE}_1.$FASTQEXT
  PAIR2=$FASTQFOLDER/${FILEBASE}_2.$FASTQEXT
  PREFIX=$STRAIN
  # END THIS PART IS PROBABLY PROJECT SPECIFIC
  echo "STRAIN is $STRAIN $PAIR1 $PAIR2"

  TMPBAMFILE=$TMPOUTDIR/$STRAIN.unsrt.bam
  SRTED=$TMPOUTDIR/$STRAIN.srt.bam
  DDFILE=$TMPOUTDIR/$STRAIN.DD.bam
  FINALFILE=$ALNFOLDER/$STRAIN.$HTCEXT
```

```
  READGROUP="@RG\tID:$STRAIN\tSM:$STRAIN\tLB:$PREFIX\tPL:illumina\tCN:$RGCENTER"

  if [ ! -s $FINALFILE ]; then
    if [ ! -s $DDFILE ]; then
      if [ ! -s $SRTED ]; then
        if [ -e $PAIR1 ]; then
          if [ ! -f $TMPBAMFILE ]; then
            # potential switch this to bwa-mem2 for extra speed
            bwa mem -t $CPU -R $READGROUP $REFGENOME $PAIR1 $PAIR2 | samtools view -1 -o $TMPBAMFILE
          fi
        else
          echo "Cannot find $PAIR1, skipping $STRAIN"
          exit
        fi
        samtools fixmate --threads $CPU -O bam $TMPBAMFILE $TEMP/${STRAIN}.fixmate.bam
        samtools sort --threads $CPU -O bam -o $SRTED -T $TEMP $TEMP/${STRAIN}.fixmate.bam
        if [ -f $SRTED ]; then
          rm -f $TEMP/${STRAIN}.fixmate.bam $TMPBAMFILE
        fi
      fi # SRTED file exists or was created by this block

      time java -jar $PICARD MarkDuplicates I=$SRTED O=$DDFILE \
        METRICS_FILE=logs/$STRAIN.dedup.metrics CREATE_INDEX=true VALIDATION_STRINGENCY=SILENT
      if [ -f $DDFILE ]; then
        rm -f $SRTED
      fi
    fi # DDFILE is created after this or already exists

    samtools view -O $HTCFORMAT --threads $CPU --reference $REFGENOME -o $FINALFILE $DDFILE
    samtools index $FINALFILE

    if [ -f $FINALFILE ]; then
      rm -f $DDFILE
      rm -f $(echo $DDFILE | sed 's/bam$/bai/')
    fi
  fi #FINALFILE created or already exists


  # The rest of this could be skipped as it is for a project to extract the UNMAPPED reads and assemble t
  FQ=$(basename $FASTQEXT .gz)
  UMAP=$UNMAPPED/${STRAIN}.$FQ
  UMAPSINGLE=$UNMAPPED/${STRAIN}_single.$FQ
  #echo "$UMAP $UMAPSINGLE $FQ"

  if [ ! -f $UMAP ]; then
    module load BBMap
    samtools fastq -f 4 --threads $CPU -N -s $UMAPSINGLE -o $UMAP $FINALFILE
    pigz $UMAPSINGLE
    repair.sh in=$UMAP out=$UMAP.gz
    unlink $UMAP
  fi
done
```

Step 3. This step converts the .cram files (which are BAM files but in a more compressed format) into g.vcf files which are for calling all possible variants. You run it again with array jobs and one job per strain.

```
sbatch --array=1-4 pipeline_GATK/02_call_gvcf.sh
```

```
#!/bin/bash -l
#SBATCH -p epyc -N 1 -n 1 -c 16 --mem 32gb --time 48:00:00
#SBATCH -J make_gvcf
#SBATCH -o logs/%x.%A_%a.log

module load picard
module load java
module load gatk/4    # GATK4 syntax
module load bcftools

MEM=32g
SAMPFILE=samples.csv

if [ -f config.txt ]; then
    source config.txt
fi

DICT=$(echo $REFGENOME | sed 's/fasta$/dict/')

if [ ! -f $DICT ]; then
    picard CreateSequenceDictionary R=$REFGENOME O=$DICT
fi
mkdir -p $VARIANTFOLDER
CPU=${SLURM_CPUS_PER_TASK:-1}
N=${SLURM_ARRAY_TASK_ID:-$1}

if [ -z "$N" ]; then
 echo "need to provide a number by --array slurm or on the cmdline"
 exit
fi

hostname
date
IFS=,
tail -n +2 $SAMPFILE | sed -n ${N}p | while read STRAIN SAMPID
do
  # BEGIN THIS PART IS PROJECT SPECIFIC LIKELY
  # END THIS PART IS PROJECT SPECIFIC LIKELY
  echo "STRAIN is $STRAIN"
  GVCF=$VARIANTFOLDER/$STRAIN.g.vcf
  ALNFILE=$ALNFOLDER/$STRAIN.$HTCEXT
  if [ -s $GVCF.gz ]; then
    echo "Skipping $STRAIN - Already called $STRAIN.g.vcf.gz"
    exit
  fi
  if [[ ! -f $GVCF || $ALNFILE -nt $GVCF ]]; then
      time gatk --java-options -Xmx${MEM} HaplotypeCaller \
      --emit-ref-confidence GVCF --sample-ploidy 1 \
      --input $ALNFILE --reference $REFGENOME \
      --output $GVCF --native-pair-hmm-threads $CPU \
         -G StandardAnnotation -G AS_StandardAnnotation -G StandardHCAnnotation
 fi
 bgzip --threads $CPU -f $GVCF
 tabix $GVCF.gz
```

```
done
date
```

Step 4. This step runs GVCF -> final VCF but one chromosome at a time. The number of chromosomes is the number of sequences in the genome FASTA file or you can count with `wc -l genome/*.fai`

If your genome is fragmented you will want to adjust the parameter in `config.txt` file so that `GVCF_INTERVAL=1` is more like 5 or 10 and then adjust your job number by that factor. Eg if you have 1000 contigs and `GVCF_INTERVAL=5` then you would want to run array jobs with $1000/5 = 200$ instead of 1000 jobs.

```
# 8 tasks = the 8 chromosomes in the NCBI A. fumigatus Af293 genome (with GVCF_INTERVAL=1)
sbatch --array=1-8 pipeline_GATK/03_jointGVCF_call_slice.sh
```

Here is the Code

```
#!/bin/bash -l
#SBATCH -p epyc -N 1 -n 1 -c 2 --mem 24G --time 48:00:00
#SBATCH -J slice.GVCFGeno
#SBATCH -o logs/%x.%A_%a.log
hostname
MEM=24g
module load picard
module load gatk/4    # GATK4 syntax
module load java
module load bcftools
module load parallel
module load samtools

source config.txt

declare -x TEMPDIR=$TEMP/$USER/$$

cleanup() {
    #echo "rm temp is: $TEMPDIR"
    rm -rf $TEMPDIR
    rmdir $TEMPDIR
}

# Set trap to ensure cleanup is run
trap "cleanup; rm -rf $TEMPDIR; exit" SIGHUP SIGINT SIGTERM EXIT

GVCF_INTERVAL=1
N=${SLURM_ARRAY_TASK_ID:-$1}

if [ -z "$N" ]; then
    echo "Need an array id or cmdline val for the job"
    exit
fi
if [ -f config.txt ]; then
    source config.txt
fi
if [ -z $SLICEVCF ]; then
    SLICEVCF=vcf_slice
fi
mkdir -p $SLICEVCF
STEM=$SLICEVCF/$PREFIX.$N
GENOVCFOUT=$STEM.all.vcf
FILTERSNP=$STEM.SNP.filter.vcf
```

```
FILTERINDEL=$STEM.INDEL.filter.vcf
SELECTSNP=$STEM.SNP.selected.vcf
SELECTINDEL=$STEM.INDEL.selected.vcf

if [ ! -f $REFGENOME.fai ]; then
    samtools faidx $REFGENOME
fi
NSTART=$(perl -e "printf('%d',1 + $GVCF_INTERVAL * ($N - 1))")
NEND=$(perl -e "printf('%d',$GVCF_INTERVAL * $N)")
MAX=$(wc -l $REFGENOME.fai | awk '{print $1}')
if [ "$NSTART" -gt "$MAX" ]; then
    echo "NSTART ($NSTART) > $MAX"
    exit
fi
if [ "$NEND" -gt "$MAX" ]; then
    NEND=$MAX
fi
echo "$NSTART -> $NEND"

CPU=${SLURM_CPUS_PER_TASK:-1}
if [[ $(ls $GVCFFOLDER | grep -c -P "\.g.vcf$") -gt "0" ]]; then
    parallel -j $CPU bgzip {} ::: $GVCFFOLDER/*.g.vcf
    parallel -j $CPU tabix -f {} ::: $GVCFFOLDER/*.g.vcf.gz
fi

FILES=$(ls $GVCFFOLDER/*.g.vcf.gz | sort | perl -p -e 's/(\S+)\n/-V $1 /')
INTERVALS=$(cut -f1 $REFGENOME.fai  | sed -n "${NSTART},${NEND}p" | perl -p -e 's/(\S+)\n/--intervals $1
mkdir -p $TEMPDIR
if [ ! -f $GENOVCFOUT.gz ]; then
    if [ ! -f $GENOVCFOUT ]; then
    DB=$TEMPDIR/${GVCFFOLDER}_slice_$N
    rm -rf $DB
    gatk  --java-options "-Xmx$MEM -Xms$MEM" GenomicsDBImport --consolidate --merge-input-intervals --gen
    #--reader-threads $CPU
    #gatk  --java-options "-Xmx$MEM -Xms$MEM" GenomicsDBImport --genomicsdb-workspace-path $DB $FILES $IN
    time gatk GenotypeGVCFs --reference $REFGENOME --output $GENOVCFOUT -V gendb://$DB --tmp-dir $TEMPDIR
    ls -l $TEMPDIR
    rm -rf $DB
    fi
    if [ -f $GENOVCFOUT ]; then
        bgzip $GENOVCFOUT
        tabix $GENOVCFOUT.gz
    fi
fi
TYPE=SNP
echo "VCF = $STEM.$TYPE.vcf.gz"
if [[ ! -f $STEM.$TYPE.vcf.gz ]]; then
    gatk SelectVariants \
    -R $REFGENOME \
    --variant $GENOVCFOUT.gz \
    -O $STEM.$TYPE.vcf \
    --restrict-alleles-to BIALLELIC \
    --select-type-to-include $TYPE --create-output-variant-index false

    bgzip $STEM.$TYPE.vcf
```

```
    tabix $STEM.$TYPE.vcf.gz
fi

if [[ ! -f $FILTERSNP.gz || $STEM.$TYPE.vcf.gz -nt $FILTERSNP.gz ]]; then
    gatk VariantFiltration --output $FILTERSNP \
    --variant $STEM.$TYPE.vcf.gz -R $REFGENOME \
    --cluster-window-size 10  \
    --filter-expression "QD < 2.0" --filter-name QualByDepth \
    --filter-expression "MQ < 40.0" --filter-name MapQual \
    --filter-expression "QUAL < 100" --filter-name QScore \
    --filter-expression "SOR > 4.0" --filter-name StrandOddsRatio \
    --filter-expression "FS > 60.0" --filter-name FisherStrandBias \
    --missing-values-evaluate-as-failing --create-output-variant-index false

#   --filter-expression "MQRankSum < -12.5" --filter-name MapQualityRankSum \
#   --filter-expression "ReadPosRankSum < -8.0" --filter-name ReadPosRank \

    bgzip $FILTERSNP
    tabix $FILTERSNP.gz
fi

if [[ ! -f $SELECTSNP.gz || $FILTERSNP.gz -nt $SELECTSNP.gz ]]; then
    gatk SelectVariants -R $REFGENOME \
    --variant $FILTERSNP.gz \
    --output $SELECTSNP \
    --exclude-filtered --create-output-variant-index false
    bgzip $SELECTSNP
    tabix $SELECTSNP.gz
fi

TYPE=INDEL
if [ ! -f $STEM.$TYPE.vcf.gz ]; then
    gatk SelectVariants \
        -R $REFGENOME \
        --variant $GENOVCFOUT.gz \
        -O $STEM.$TYPE.vcf  --select-type-to-include MIXED --select-type-to-include MNP \
        --select-type-to-include $TYPE --create-output-variant-index false
    bgzip $STEM.$TYPE.vcf
    tabix $STEM.$TYPE.vcf.gz
fi

if [[ ! -f $FILTERINDEL.gz || $STEM.$TYPE.vcf.gz -nt $FILTERINDEL.gz ]]; then
    gatk VariantFiltration --output $FILTERINDEL \
    --variant $STEM.$TYPE.vcf.gz -R $REFGENOME \
    --cluster-window-size 10  -filter "QD < 2.0" --filter-name QualByDepth \
    -filter "SOR > 10.0" --filter-name StrandOddsRatio \
    -filter "FS > 200.0" --filter-name FisherStrandBias \
    -filter "InbreedingCoeff < -0.8" --filter-name InbreedCoef \
    --create-output-variant-index false

#   -filter "ReadPosRankSum < -20.0" --filter-name ReadPosRank \
#   -filter "MQRankSum < -12.5" --filter-name MapQualityRankSum \

    bgzip $FILTERINDEL
    tabix $FILTERINDEL.gz
```

```
fi

if [[ ! -f $SELECTINDEL.gz || $FILTERINDEL.gz -nt $SELECTINDEL.gz ]]; then
    gatk SelectVariants -R $REFGENOME \
    --variant $FILTERINDEL.gz \
    --output $SELECTINDEL \
    --exclude-filtered --create-output-variant-index false
    bgzip $SELECTINDEL
    tabix $SELECTINDEL.gz
fi

if [ -d $TEMPDIR ]; then
    rmdir $TEMPDIR
fi
```

Step 5. Finally to combine all the slices we use this script which is very fast and combines the slices.

```
sbatch pipeline_GATK/04_combine_vcf.sh
```

Step 6. To get predictions about what the impact of SNPs are, which genes they fall in, and whether they are synonymous or non-synonymous changes.

```
sbatch pipeline_GATK/08_snpEff.sh
```

```
#!/bin/bash -l
#SBATCH -p epyc -N 1 -n 1 -c 2 --mem 64G --time 12:00:00
#SBATCH -J snpEff
#SBATCH -o logs/%x.%j.log
module load miniconda3
module load snpEff
module load bcftools
module load tabix
# THIS IS AN EXAMPLE OF HOW TO MAKE SNPEFF - it is for A.fumigatus
# SNPEFFGENOME and GFFGENOME come from config.txt

MEM=64g

# this module defines SNPEFFJAR and SNPEFFDIR
if [ -f config.txt ]; then
    source config.txt
fi
GFFGENOMEFILE=$GENOMEFOLDER/$GFFGENOME
if [ -z $SNPEFFJAR ]; then
 echo "need to defined \$SNPEFFJAR in module or config.txt"
 exit
fi
if [ -z $SNPEFFDIR ]; then
 echo "need to defined \$SNPEFFDIR in module or config.txt"
 exit
fi
# could make this a confi

if [ -z $FINALVCF ]; then
    echo "need a FINALVCF variable in config.txt"
    exit
fi

mkdir -p $SNPEFFOUT
```

```
## NOTE YOU WILL NEED TO FIX THIS FOR YOUR CUSTOM GENOME
if [ ! -e $SNPEFFOUT/$snpEffConfig ]; then
    rsync -a $SNPEFFDIR/snpEff.config $SNPEFFOUT/$snpEffConfig
    echo "# AfumAf293.ncbi " >> $SNPEFFOUT/$snpEffConfig
    # CHANGE this to your genome name and source - though this is really not important - $SNPEFFGENOME.ge
    echo "$SNPEFFGENOME.genome : Aspergillus fumigatus Af293 NCBI RefSeq" >> $SNPEFFOUT/$snpEffConfig
    chroms=$(grep '##sequence-region' $GFFGENOMEFILE | awk '{print $2}' | perl -p -e 's/\n/, /' | perl -p
    echo -e "\t$SNPEFFGENOME.chromosomes: $chroms" >> $SNPEFFOUT/$snpEffConfig
    # If your genome has a mitochondrial contig, give it the right codon table, e.g.
    # echo -e "\t$SNPEFFGENOME.MITO_CONTIG_NAME.codonTable : Mold_Mitochondrial" >> $SNPEFFOUT/$snpEffCon
    # (the NCBI RefSeq Af293 assembly has only the 8 nuclear chromosomes)
    mkdir -p $SNPEFFOUT/data/$SNPEFFGENOME
    gzip -c $GFFGENOMEFILE > $SNPEFFOUT/data/$SNPEFFGENOME/genes.gff.gz
    rsync -aL $REFGENOME $SNPEFFOUT/data/$SNPEFFGENOME/sequences.fa

    java -Xmx$MEM -jar $SNPEFFJAR build -dataDir `pwd`/$SNPEFFOUT/data -c $SNPEFFOUT/$snpEffConfig -gff3
fi
pushd $SNPEFFOUT
COMBVCF="../$FINALVCF/$PREFIX.SNP.combined_selected.vcf.gz ../$FINALVCF/$PREFIX.INDEL.combined_selected.v
for n in $COMBVCF
do
 echo $n
 st=$(echo $n | perl -p -e 's/\.gz//')
 if [ ! -f $n ]; then
     bgzip $st
 fi
 if [ ! -f $n.tbi ]; then
     tabix $n
 fi
done
INVCF=$PREFIX.allvariants_combined_selected.vcf
OUTVCF=$PREFIX.snpEff.vcf
OUTTAB=$PREFIX.snpEff.tab
OUTMATRIX=$PREFIX.snpEff.matrix.tsv
DOMAINVAR=$PREFIX.snpEff.domain_variant.tsv
bcftools concat -a -d both -o $INVCF -O v $COMBVCF
java -Xmx$MEM -jar $SNPEFFJAR eff -dataDir `pwd`/data -v $SNPEFFGENOME $INVCF > $OUTVCF

bcftools query -H -f '%CHROM\t%POS\t%REF\t%ALT{0}[\t%TGT]\t%INFO/ANN\n' $OUTVCF > $OUTTAB

# Optional extra steps from the original research pipeline (helper scripts not included here):
# map variants onto InterPro protein domains, and make a gene x strain matrix
# ../scripts/map_snpEff2domains.py --vcf $OUTVCF --domains DOMAINS.txt --output $DOMAINVAR
# ../scripts/snpEff_2_tab.py $OUTVCF > $OUTMATRIX
# The $OUTTAB table from bcftools query above has the same information; the
# SNP workshop (Workshop_SNPs) shows how to summarize tables like it in Python.
```