BAM
VCF

reference ← chromosome
plasmid1
⋮

2.6.6 Yan
Jamie
Errol

```
#!/usr/local/bin/python
#from var.flt.vcf to construct SNP position list; from reads.pileup to extract the nucleotide
base at each SNP position for each sample to construct the SNP fasta file. Multiple threads.

from Bio import SeqIO
from optparse import OptionParser
import sys,string,os,shutil
import re
import operator
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from os.path import join
from operator import itemgetter
import subprocess
from subprocess import call
from datetime import datetime
import threading
import time

class FuncThread(threading.Thread):
    def __init__(self,target,*args):
        self._target = target
        self._args = args
        threading.Thread.__init__(self)

    def run(self):
        self._target(*self._args)
```

manage multiple threading for pileup using SAMTOOLS

```
###from the pileup file, call the base for each SNP position.

def get_value_from_data(base,length,data):
    ret = ""
    charHash = dict()
    charHash[".,"] = 0
    charHash["A"] = 0
    charHash["C"] = 0
    charHash["T"] = 0
    charHash["G"] = 0
    charHash["N"] = 0

    i = 0
    while i < len(data):
        char = data[i]
        if char == '.' or char == ',':
            charHash[".,"] += 1
        elif char == 'A' or char == 'a':
            charHash["A"] += 1
        elif char == 'C' or char == 'c':
            charHash["C"] += 1
        elif char == 'T' or char == 't':
            charHash["T"] += 1
        elif char == 'G' or char == 'g':
            charHash["G"] += 1
```

get consensus call from pileup ("reads.pileup")
calculates # of each symbol

```
        elif char == 'N' or char == 'n':
            charHash["N"] += 1
        elif char == '+' or char == '-':
            countStr = ""
            count = 1
            while re.match("\d" ,data[i + 1]):
                countStr += data[i + 1]
                i += 1
#            print countStr
            if countStr != "":
                count = int(countStr)
            i += count
        elif char == '^':
            if data[i+1] != "." and data[i+1] != ',':
                i +=1
        i += 1
```

Produces a HashMap of bases for Isolates

```
    ret = max(charHash.iteritems(), key=operator.itemgetter(1))[0]
    if charHash[ret] <= (int(length)/2):
        ret = "-"
    elif ret ==".,":
        ret = base
    return ret
```

gap if nothing > 50%

```
###store each pileup information to a Hash.
def createPositionValueHash(pileupFilePath):
    positionValueHash = dict()
    pileupFile = open(pileupFilePath, "r")
    while 1:
        curpileupFileLine = pileupFile.readline()
        if not curpileupFileLine:
            break
        seqString = ""
        curLineData = curpileupFileLine.split()
        if len(curLineData) <5:
            continue
        positionValueHash[curLineData[0] + ":" + curLineData[1]] = get_value_from_data(
        curLineData[2],curLineData[3],curLineData[4])
    pileupFile.close()
    return positionValueHash
```

read each line

read in pileup,

'id', chromosome
plasmid1
⋮

location in

get nucleotide for ~~#~~ that position

```
def pileup(filePath,snplistFilePath,dirName):
    seqString = ""
    os.chdir(filePath)

    ####generate pileup files, using snplist file and the reference fasta file.
    subprocess.call("samtools mpileup -l " + opts.mainPath + opts.snplistFileName + " -f " +
    opts.mainPath + opts.Reference + " reads.bam > reads.pileup",shell=True )
```

from bamfile

```
    ####read in pileup file and store information to a hash
    positionValueHash = createPositionValueHash(filePath + "/reads.pileup")

    ####append the nucleotide to the record
    snplistFile_r = open(snplistFilePath, "r")
```

seqs in one fasta file

2
0

```
    snplistFile_r.seek(0)
    i = 0
    while 1:
        curSnplistLine = snplistFile_r.readline()
        if not curSnplistLine:
            break
        i = i+1
        curSnplistData = curSnplistLine.split()
        if  len(curSnplistData) <2:
            print "snplistfile: bad line# "+i+" line="+curSnplistLine
            continue
        chrom = curSnplistData[0]
        pos = curSnplistData[1]

        if positionValueHash.has_key(chrom + ":" + pos):
            seqString += positionValueHash[chrom + ":" + pos]
        else:
            seqString += "-"
    print "length of seqRecordString="+str(len(seqString))
    seq = Seq(seqString)
    seqRecord = SeqRecord(seq,id=dirName)
    records.append(seqRecord)
    snplistFile_r.close()
```

threads

```
#### Command line usage
usage = "usage: %prog -n 10 -d /home/yan.luo/Desktop/ -f path.txt -r reference -l snplist.txt
-a snpma.fasta"

p = OptionParser(usage)
p.add_option ("-n","--cpu",dest="maxThread",type="int",default=15,help="Max count of cocurrent
thread (default=15)")
p.add_option ("-d","--mainPath",dest="mainPath",default=
"/home/yan.luo/Desktop/analysis/Montevideo/XL-C2/bowtie/Matrices/",help="Path for all files")
p.add_option ("-r","--Reference",dest="Reference",default="CFSAN001339_pacbio.fasta",help=
"reference for mapping")
p.add_option ("-f","--pathFileName",dest="pathFileName",default="path.txt",help="Path file name")
p.add_option ("-l","--snplistFileName",dest="snplistFileName",default="snplist.txt",help=
"Snplist file name")
p.add_option ("-a","--snpmaFileName",dest="snpmaFileName",default="snpma.fa",help="fasta file
name")
(opts,args)=p.parse_args()


pathFile = open(opts.mainPath + opts.pathFileName, "r")
snplistFile = open(opts.mainPath + opts.snplistFileName, "w")
snplistHash = dict()

###read all *vcf file for SNP list

while 1:
    filePath = pathFile.readline()[:-1]
    dirName = filePath.split(os.sep)[-1]
    if not filePath:
```

```
        break
    print filePath
    print dirName
    vcfFile = open(filePath + "/var.flt.vcf","r")
    while 1:
        curVcfFileLine = vcfFile.readline()
        if not curVcfFileLine:
            break
        if curVcfFileLine.startswith("#"):
            continue

        curLineData = curVcfFileLine.split()
        chrom = curLineData[0]
        pos = curLineData[1]
        info = curLineData[7]
        if str("INDEL") in curLineData[7]:
            continue
        infoFields = info.split(";")
        dpFlag = False
        af1Flag = False
        for infoField in infoFields:
            infoPair = infoField.split("=")
            if infoPair[0] == "DP" and int(infoPair[1]) >= 10:
                dpFlag = True
            elif (infoPair[0] == "AF1" and infoPair[1] == "1" ) or (infoPair[0] == "AR" and
            infoPair[1] == "1.00"):
                af1Flag = True
  # find a good record fo SNP position, save data to hash
        if dpFlag and af1Flag:
            if not snplistHash.has_key(chrom + "\t" + pos):
                record = [1]
                record.append(dirName)
                snplistHash[chrom + "\t" + pos] = record
            else:
                record = snplistHash[chrom + "\t" + pos]
                record[0] += 1
                record.append(dirName)
    vcfFile.close()

for key in sorted(snplistHash.iterkeys()):
    snplistFile.write(key)
    values = snplistHash[key]
    for value in values:
        snplistFile.write("\t" + str(value))
    snplistFile.write("\n")
snplistFile.close()


pathFile.seek(0)
snplistFilePath = opts.mainPath + opts.snplistFileName
fastaFile = open(opts.mainPath + opts.snpmaFileName, "w")

records = []
threads = []
```

```
while 1:
    filePath = pathFile.readline()[:-1]
    dirName = filePath.split(os.sep)[-1]
    if not filePath:
        break

    t1 = FuncThread(pileup,filePath,snplistFilePath,dirName)
    threads.append(t1)
    while threading.activeCount() > opts.maxThread:
        time.sleep(15)
    t1.start()

for thread in threads:
    thread.join()

####write the records to fasta file
SeqIO.write(records, fastaFile, "fasta")
fastaFile.close()
```

#threads