

GATK run report

Vikas Gupta, Niraj Shah and Stig U. Andersen

September 23, 2013

Contents

1. Introduction	2
2. WorkFlow	2
2..1 Fastq Mapping	2
2..2 Duplicate Marking	3
2..3 Realiging the reads	3
2..4 Unified genotyper	3
2..5 Base Recalibrator	3
3. Callable loci	4
4. SNP distribution	5
5. Heterozygosity distribution	6
5..1 Mapping on Lotus Genome	7
5..2 Average coverage on Lotus Genome	8
5..3 Inbreeding coefficient	9
6. Parsing VCF	9
6..1 Heterozygosity	10
6..2 Marker filtering criteria	11
6..2..1 Counting filtering categories	11
6..2..2 Average QUAL value	11
6..2..3 MG20 homozygous positions	11
6..2..4 Variant annotation	11
7. Python Script	12

GATK is an acronym for Genome analysis toolkit, a detailed documentation can be found at <http://www.broadinstitute.org/gatk/guide/>. GATK pipeline can be used for detecting variants from multi-sample data. One of the problems with variant detection is the misaligned reads, this issue has been addressed in GATK by considering the per-base quality score which represents the probability that the base called in the read is true sequenced base. These per-base quality scores are quite inaccurate (Mark A DePristo et.al 2011) and depends on the experimental setup and technology used. These errors are eliminated from the GATK using empirically accurate per-base error model.

1. Introduction

Here we have utilized sequencing data from MG20, Gifu, Burtii and 28 additional accessions. Using this data we aim to annotate the genetic variants such as SNPs and Indels in the *Lotus japonicus*.

2. WorkFlow

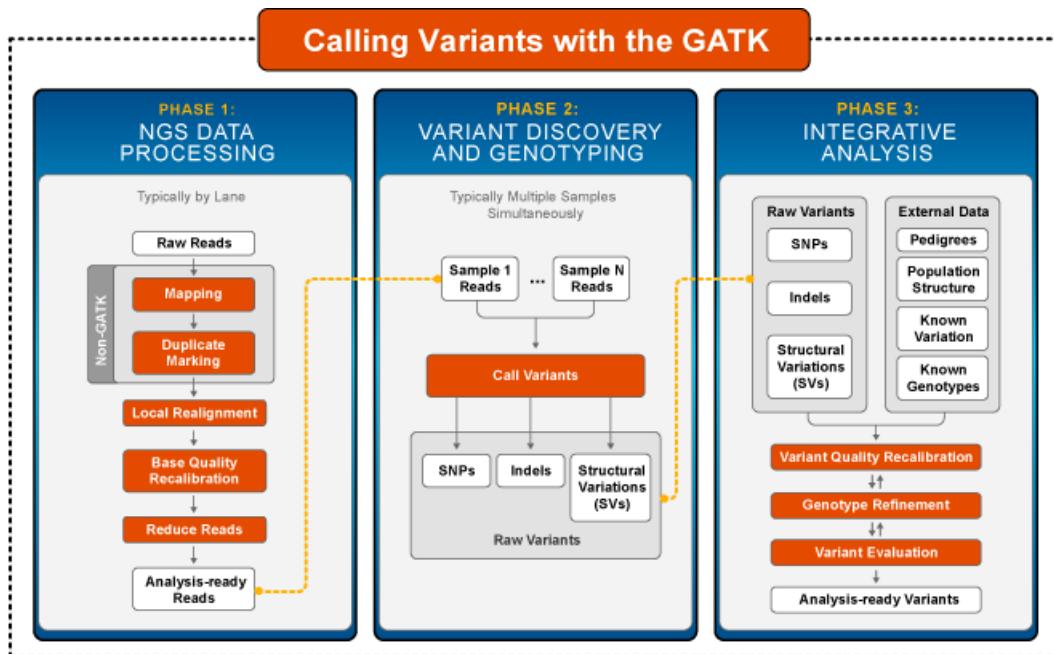


Figure 1: Three sections of GATK pipeline

We have divided work here into four major points:

1. Initial read mapping
2. Local realignment around indels
3. Base quality score recalibration
4. SNP and indel detection

2.1 Fastq Mapping

Fastq files were mapped to the Lotus genome 3.0 using the BWA version-0.7.5a-r405 with default parameters and with appropriate insert size for the pair-end libraries. GATK pipeline does not support files without read groups so these were added whenever necessary with Picard *AddOrReplaceReadgroup* function. All the mapped files are placed at genome.au.dk. /home/vgupta/LotusGenome/100_data/01_Jin_BamFile/02_withReadGroup.

2.2 Duplicate Marking

Duplicates were filtered using Picard toolkit version 1.96. Two things to remember when using Picard for duplicate filtering:

1. Set `VALIDATION_STRINGENCY=LENIENT` otherwise Picard will not accept unmapped read positions.
2. Use `TMP_DIR` variable to a folder where you have sufficient space.

2.3 Realiging the reads

Duplicate filtered reads are mapped back on the genome using `RealignerTargetCreator` and `IndelRealigner` commands from the GATK. All the fastq files must follow the sanger quality encoding. `IndelRealigner` locally aligns the reads to minimize the mismatches. Also many reads are misaligned due to presence of insertions and deletions, leading to many false discoveries of SNPs.

2.4 Unified genotyper

Realigned bam files are parsed through the unified genotyper to procure a primary list of snp and indel list. `-glm BOTH` option must be used to predict indels together with SNPs. A higher degree of calling confidence cut-off can be used to insure minimal false positives. Unified genotyper uses a Bayesian genotype likelihood model to find genotypes and allele frequency in a population of N samples. It provides a posterior probability of there being a segregating variant allele at each locus as well as for the genotype of each sample.

2.5 Base Recalibrator

In this step, base quality scores are recalibrated. Given a set of high confidence SNPs from the earlier step, program calculates an empirical probability of error given the particular covariates seen at this site, where $p(\text{error}) = \text{num mismatches} / \text{num observations}$.

3. Callable loci

Genome-wide Callable loci distribution

4. SNP distribution

Genome-wide SNP density distribution

/Users/vgupta/Desktop/03_Lotus_annotation/2013_week35/snpDens

Figure 2: SNP distribution

5. Heterozygosity distribution

Genome-wide heterozygosity distribution

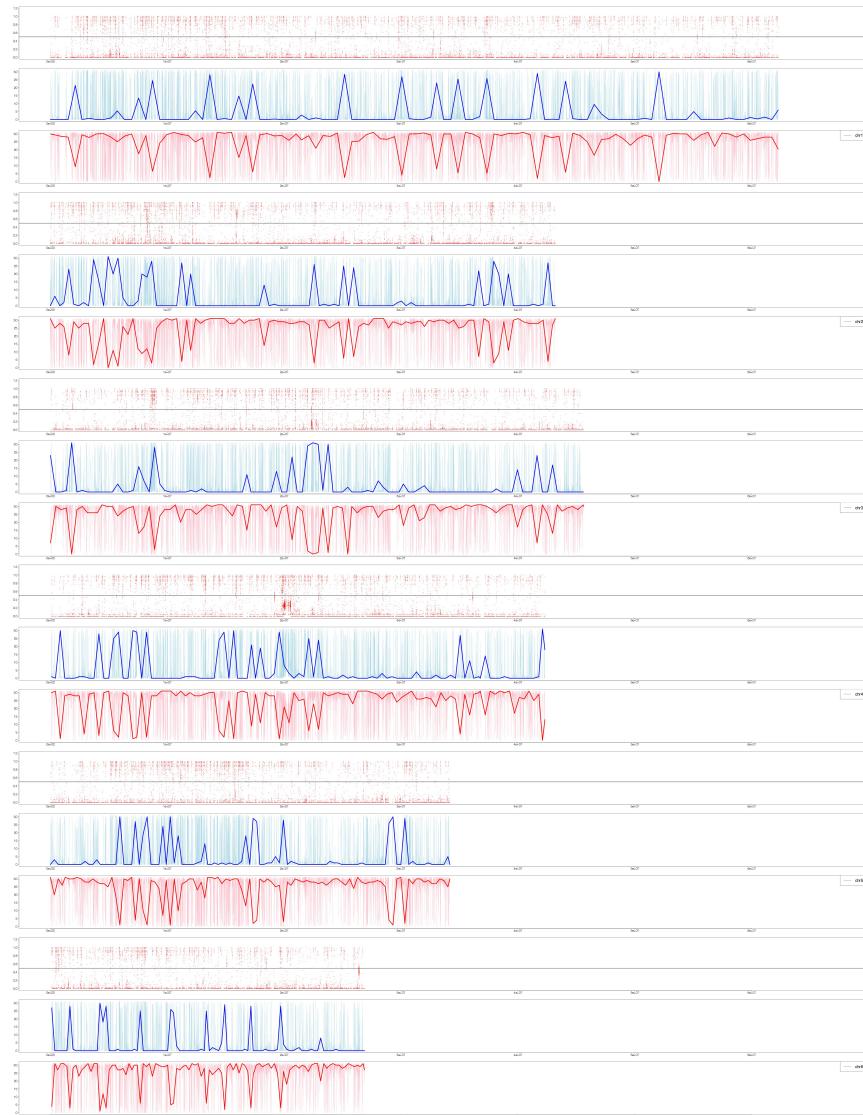


Figure 3: Heterozygosity distribution

5.1 Mapping on Lotus Genome

BamFile	of reads	Reads Mapped	Reads Unmapped	Perc. of reads mapped
mg004.bam.withRG.bam	32995084	29099069	3896015	88
mg010.bam.withRG.bam	33247344	30122887	3124457	90
mg012.bam.withRG.bam	22359352	19525310	2834042	87
mg019.bam.withRG.bam	35052674	30390626	4662048	86
mg023.bam.withRG.bam	36685962	33540205	3145757	91
mg036.bam.withRG.bam	32553458	29679491	2873967	91
mg049.bam.withRG.bam	34495526	31538761	2956765	91
mg051.bam.withRG.bam	35242522	31626258	3616264	89
mg062.bam.withRG.bam	33199860	30109966	3089894	90
mg072.bam.withRG.bam	29408018	26441623	2966395	89
mg073.bam.withRG.bam	36264818	33322832	2941986	91
mg077.bam.withRG.bam	31920060	28848565	3071495	90
mg080.bam.withRG.bam	36617306	32848091	3769215	89
mg082.bam.withRG.bam	36476482	33318768	3157714	91
mg083.bam.withRG.bam	25010356	22124538	2885818	88
mg086.bam.withRG.bam	35101822	31854856	3246966	90
mg089.bam.withRG.bam	34011174	30543091	3468083	89
mg093.bam.withRG.bam	29660014	26868546	2791468	90
mg095.bam.withRG.bam	26555582	23835904	2719678	89
mg097.bam.withRG.bam	33749516	30514362	3235154	90
mg101.bam.withRG.bam	47140196	42698502	4441694	90
mg107.bam.withRG.bam	43189472	39947508	3241964	92
mg109.bam.withRG.bam	30911626	27816443	3095183	89
mg112.bam.withRG.bam	45126622	41594055	3532567	92
mg113.bam.withRG.bam	33405764	29292248	4113516	87
mg118.bam.withRG.bam	32252874	29096722	3156152	90
mg123.bam.withRG.bam	21800012	19070543	2729469	87
mg128.bam.withRG.bam	33888258	30600839	3287419	90

Table 1: Reads mapped on Lotus genome

5.2 Average coverage on Lotus Genome

FileName	AverageCoverage	StdDev
MG20	51.4679	137.669
Gifu	80.7166	163.611
mg004	7.12603	38.2242
mg010	7.89245	45.8791
mg012	5.15874	31.085
mg019	8.33885	39.0382
mg023	8.88417	51.2091
mg036	9.49761	43.2652
mg049	9.33341	47.7517
mg051	8.35398	55.9712
mg062	7.66547	55.2581
mg072	6.96558	49.5173
mg073	7.84491	40.8285
mg077	6.85289	48.9573
mg080	7.74727	41.1546
mg082	7.99317	35.8655
mg083	5.75298	30.0052
mg086	7.50583	33.8896
mg089	7.30731	38.3547
mg093	6.58203	29.0308
mg095	NULL	NULL
mg097	7.11818	37.69
mg101	9.87511	55.1488
mg107	9.3095	47.0785
mg109	6.96913	35.0317
mg112	9.66309	39.9193
mg113	7.14424	48.0236
mg118	7.18516	49.812
mg123	5.03514	40.5754

Table 2: Average Coverage of re-sequenced lines

5.3 Inbreeding coefficient

A good example of calculating the F (Inbreeding co-efficient) is explained at <http://www.uwyo.edu/dbmcd/popecol/maylects/fst.html>. Hardy-Weinberg Equilibrium calculates values of p and q based on the dominant alleles and using p and q values calculates expected genotype counts. Based on the differences on genotypic counts, we can categorize a population as perfect fit, homozygotes excessive (inbred) or homozygotes deficient (isolate breaking).

6. Parsing VCF

Note: Total depth on a given position doesn't add up to sum of individual sample read depth. Explanation is copied from http://www.broadinstitute.org/gatk/gatkdocs/org_broadinstitute_sting_gatk_walkers_annotation_Coverage.html.

While the sample-level (FORMAT) DP field describes the total depth of reads that passed the caller's internal quality control metrics (like MAPQ > 17, for example), the INFO field DP represents the unfiltered depth over all samples. Note though that the DP is affected by downsampling (-dcov), so the max value one can obtain for N samples with -dcov D is N * D.

6.1 Heterozygosity

We have measured the level of heterozygosity using the final vcf file produced by GATK. It reports a total of 394,861,527 base-pairs which includes all the callable sites. 5,280,517 markers(SNPs and Indels) were detected. Expect a small fraction of 40,000, all 1.39 million makers from prior analysis were recovered. MG20 echotype has been inbreed for many generations and expected to be highly homozygous but due to error in the mapping and assembly, we found a high fraction of markers were heterozygous across the genome. To remove these potentially false positive markers, we filtered all the markers where genotype call was not 0/0 (reference based homozygous) for the MG20 resulted into a set of 3,989,842 markers among MG20, Gifu, Burtii and 28 re-sequenced. Lines in the analysis seems to highly homozygous hence can be analysed as haploid.

Sample	HetCount	HomoCount	HetPer	HomoPer
MG20	0	3989842	0.00	1.00
Gifu	398844	3529289	0.10	0.90
Burtii	600378	3138281	0.16	0.84
mg004	394300	2980909	0.12	0.88
mg010	299897	2869762	0.09	0.91
mg012	325414	2806793	0.10	0.90
mg019	288101	2756945	0.09	0.91
mg023	319694	2844582	0.10	0.90
mg036	224498	2417182	0.08	0.92
mg049	255672	2610175	0.09	0.91
mg051	400899	2752953	0.13	0.87
mg062	328410	2895575	0.10	0.90
mg072	296329	2839960	0.09	0.91
mg073	388106	3111348	0.11	0.89
mg077	367029	3075315	0.11	0.89
mg080	417848	3071118	0.12	0.88
mg082	365037	3067565	0.11	0.89
mg083	334598	2843448	0.11	0.89
mg086	388324	3099269	0.11	0.89
mg089	396557	3046736	0.12	0.88
mg093	356494	3000176	0.11	0.89
mg095	355525	2970284	0.11	0.89
mg097	406711	3107906	0.12	0.88
mg101	417805	3136307	0.12	0.88
mg107	398167	3131309	0.11	0.89
mg109	353309	2944567	0.11	0.89
mg112	390936	3152211	0.11	0.89
mg113	395130	2980594	0.12	0.88
mg118	364239	2964873	0.11	0.89
mg123	315600	2810258	0.10	0.90
mg128	377147	3059781	0.11	0.89

Table 3: Heterozygosity of all the lines

6.2 Marker filtering criteria

Low quality mappings should be filtered before we consolidate a high confident marker list. We need a filtering criteria to remove false positive markers, similar filtering stradegy must be used for also considering the callable fraction of the genome. To startwith we are considering three parameters for the filtering, QUAL (phred-scaled quality score) values, FILTER categories and the MQ (RMS mapping quality).

6.2.1 Counting filtering categories

```
[vgupta@fe1 02_withReadGroup]cut -f7 20130905.snp.vcf| sort| uniq -c  
228,849,923 .  
166,011,565 LowQual
```

6..2.2 Average QUAL value

```
[vgupta@s01n14 02_withReadGroup]grep -v'"20130905.snp.vcf|awk'if(min == "")min = max =6; if((6) >  
max)max = (6); if((6) < min)min =6; total+=(6); count+= 1ENDprinttotal, total/count, min, max'  
Total : 3.86946e + 12Average : 9,799.54Min : .Max : 34,359,763,362.97
```

6.2.3 MG20 homozygous positions

A total of 394,861,527 positions were reported using Unified Genotyper, of which, 307,385,387 positions were homozygous for MG20. We also filtered the vcf file for all the positions where MG20 was the only sample with the coverage, resulting positions will be refered as Callable Loci which consist a total of 301,306,971 genomic positions.

6.2.4 Variant annotation

Using SNPeff http://snpeff.sourceforge.net/SnpEff_manual.html, we have annotated 1,354,324 snps and 234,209 indels. VCF files for respective data are as following:

```
/u/pm/data/2013_09_11_lotus_snps/20130905.snp.vcf.markers.snps.inbreed.ref  
/u/pm/data/2013_09_11_lotus_snps/20130905.snp.vcf.markers.indels.inbreed.ref
```

7. Python Script

Listing 1: GATK pipeline

```
1 #-----+  
2 #  
3 # 116_runGATK.py - script to run GATK analysis  
4 #  
5 #-----+  
6 #  
7 # AUTHOR: Vikas Gupta  
8 # CONTACT: vikas0633@gmail.com  
9 # STARTED: 09/06/2013  
10 # UPDATED: 09/06/2013  
11 #  
12 # DESCRIPTION:  
13 #  
14 # LICENSE:  
15 # GNU General Public License, Version 3  
16 # http://www.gnu.org/licenses/gpl.html  
17 #  
18 #-----+  
19  
20 # Example:  
21 # python ~/script/python/116_runGATK.py -i 02_Stegodyphous_cdna.refined.fa.orf.  
# tr_longest_frame  
22  
23  
24 ### import modules  
25 import os,sys,getopt, re  
26  
27  
28 ### global variables  
29 global bams, ref, picard, gatk, threads, variant, sort_bams, tmp_dir  
30  
31 ### make a logfile  
32 import datetime  
33 now = datetime.datetime.now()  
34 o = open(str(now.strftime("%Y-%m-%d_%H%M"))+'logfile','w')  
35  
36  
37  
38 ### write logfile  
39  
40 def logfile(infile):  
41     o.write("Program used: \t\t%s" % "116_runGATK.py"+"\n")  
42     o.write("Program was run at: \t%s" % str(now.strftime("%Y-%m-%d_%H%M"))+"\n")  
43     o.write("Infile used: \t\t%s" % infile+"\n")  
44  
45  
46 def help():  
47     print ''  
48     python 116_runGATK.py  
49             -b <bams> [One bam file per sample seperated by  
#                 commas]  
#             -r <ref> [Reference sequence]  
#             -p <picard> [Path to picard folder MarkDuplicates.  
#                         jar]
```

```

52                         -g <gatk> [Path to GATK folder containing
53                                         GenomeAnalysisTK.jar]
54                         -t <threads> [Number of threads to be used]
55
56             '''
57             sys.exit(2)
58
59     ### main argument to
60
61     def options(argv):
62
63         global bams, ref, picard, gatk, threads, variant, sort_bams, tmp_dir
64
65         bams = ''
66         ref = ''
67         picard = ''
68         gatk = ''
69         threads = str(1)
70         variant = ''
71         sort_bams = False
72         tmp_dir = "/home/vgupta/temp"
73
74         try:
75             opts, args = getopt.getopt(argv, "hb:r:p:g:t:v:s",
76                                         ["bams=", "ref=", "picard=",
77                                         "gatk=", "threads=", "variant=", "sort_bams="])
78         except getopt.GetoptError:
79             help()
80
81         for opt, arg in opts:
82             if opt == '-h':
83                 help()
84             elif opt in ("-b", "--bams"):
85                 bams = arg
86             elif opt in ("-r", "--ref"):
87                 ref = arg
88             elif opt in ("-p", "--picard"):
89                 picard = arg
90             elif opt in ("-g", "--gatk"):
91                 gatk = arg
92             elif opt in ("-t", "--threads"):
93                 threads = str(arg)
94             elif opt in ("-v", "--variant"):
95                 variant = arg
96             elif opt in ("-s", "--sort"):
97                 sort_bams = True
98
99         logfile(bams)
100
101     def Index():
102         if not os.path.isfile(ref + '.fai'):
103             os.system('nice -n 19 samtools faidx ' + ref)
104
105     def files():
106         files = []
107         print bams
108         for file in bams.split(','):
109             print file

```

```

109         files.append(file.strip())
110     return files
111
112 def sortBams(file_list):
113     if sort_bams == True:
114         file_list_bams = []
115         for file in file_list:
116             os.system('samtools sort '+file+' '+file+'_sorted')
117             file_list_bams.append(file+'_sorted.bam')
118         return file_list_bams
119
120     return file_list
121
122 def MarkDuplicates(file_list):
123     for file in file_list:
124         print 'Marking duplicates for', file
125         os.system('java -Xmx50g -jar '+picard+'/MarkDuplicates.jar
126                         VALIDATION_STRINGENCY=LENIENT TMP_DIR='+tmp_dir +' INPUT=' + file +
127                         OUTPUT=' + file+'.dedup.bam' + METRICS_FILE=' + file + '.dups')
128
129 def ReAlign(file_list):
130     for file in file_list:
131         print 'Realigning', file
132         os.system('java -jar -Djava.io.tmpdir=' + tmp_dir + '+gatk+/GenomeAnalysisTK
133                         .jar --fix_misencoded_quality_scores -fixMisencodedQuals -U -T
134                         RealignerTargetCreator '+' -I ' + file+'.dedup.bam' + '-nt ' + threads +
135                         ' -R ' + ref + '-o ' + file+'.intervals')
136         os.system('java -jar -Djava.io.tmpdir=' + tmp_dir + '+gatk+/GenomeAnalysisTK
137                         .jar --fix_misencoded_quality_scores -fixMisencodedQuals -U -T
138                         IndelRealigner ' + '-targetIntervals ' + file+'.intervals '+' -I ' +
139                         file+'.dedup.bam' + '-R ' + ref \
140                         + ' -o ' + file+'.realigned.bam')
141
142
143 def UnifiedGenotyper(file_list):
144     if variant == '':
145         in_string = ''
146         ### make input string
147         for file in file_list:
148             os.system('samtools index '+file)
149             in_string += ' -I ' + file
150         print 'Running UnifiedGenotyper'
151         '''
152         ### for sample
153         os.system(' java -jar '+gatk+/GenomeAnalysisTK.jar ' \
154             + ' -R ' + ref \
155             + ' -T UnifiedGenotyper ' \
156             + in_string \
157             + ' -nt ' + threads \
158             + ' -o snps.90.raw.vcf ' \
159             + '-stand_call_conf 20 ' \
160             + '-stand_emit_conf 10.0 ' \
161             + '-dcov 2 ')
162
163         '''
164         os.system(' java -jar -Djava.io.tmpdir=' + tmp_dir + '+gatk+/
165                         GenomeAnalysisTK.jar ' \
166                         + ' -R ' + ref \

```

```

159     + ' -T UnifiedGenotyper -glm BOTH '\
160     + in_string \
161     + ' -nt ' + threads \
162     + ' -o ' + file + '.90.vcf' \
163     + '-stand_call_conf 90' \
164     + '-stand_emit_conf 10.0' \
165     + '-dcov 200')
166
167
168 def recal(file_list):
169     global variant
170     if variant == '':
171         variant = file_list[-1] + '.90.vcf'
172     for file in file_list:
173         print 'Running BaseRecalibrator for ', file
174         os.system('java -jar -Djava.io.tmpdir=' + tmp_dir + gatk + '/GenomeAnalysisTK.jar -U -T BaseRecalibrator -rf BadCigar ' + '-knownSites ' + variant +
175             ' -I ' + file + '.realigned.bam' + ' -R ' + ref \
176             + ' -o ' + file + '.recal.table')
177         os.system('java -jar -Djava.io.tmpdir=' + tmp_dir + gatk + '/GenomeAnalysisTK.jar -T PrintReads -R ' + ref + ' -I ' + file + '.realigned.bam -L 20 ' + ' -BQSR ' + file + '.recal.table' + ' -o ' + file + '.recal_reads.bam' )
178
179 def ReduceReads(file_list):
180     for file in file_list:
181         print 'Running ReduceReads for ', file
182         os.system('java -jar -Djava.io.tmpdir=' + tmp_dir + gatk + '/GenomeAnalysisTK.jar -U -T ReduceReads -rf BadCigar -I ' + file + '.recal_reads.bam' + ' -R ' + ref \
183             + ' -o ' + file + '.reduced.bam')
184
185 def ReUnifiedGenotyper(file_list):
186     print 'Running ReUnifiedGenotyper'
187     in_string = ''
188     ### make input string
189     for file in file_list:
190         os.system('samtools index ' + file + '.reduced.bam')
191         in_string += ' -I ' + file + '.reduced.bam'
192
193     os.system(' java -jar -Djava.io.tmpdir=' + tmp_dir + gatk + '/GenomeAnalysisTK.jar ' \
194             + ' -R ' + ref \
195             + ' -T UnifiedGenotyper -glm BOTH' \
196             + in_string \
197             + ' -nt ' + threads \
198             + ' -o snps.raw.vcf ')
199
200 def BuildErrorModelWithVQSR(file , var):
201     os.system('java -jar ' + gatk + '/GenomeAnalysisTK.jar ' \
202             + ' -T VariantRecalibrator ' \
203             + ' -R ' + ref \
204             + ' -input ' + file \
205             + ' -recalFile output.recal' \
206             + ' -tranchesFile output.tranches' \
207             + ' -nt ' + threads \
208             + ' -mode ' + var)
209
210 if __name__ == "__main__":

```

```

211     options(sys.argv[1:])
212
213     ### check if index exists
214     Index()
215
216
217     ### return the list of the bam files
218     file_list = files()
219
220     ### sort Bams file
221     file_list = sortBams(file_list)
222
223     ### mark duplicates
224     MarkDuplicates(file_list)
225
226     ### realign the reads
227     ReAlign(file_list)
228
229     ### call UnifiedGenotyper to make a primary list of variants
230     UnifiedGenotyper(file_list)
231
232     ### Baserecalibration
233     recal(file_list)
234
235     ### reducing BAM files
236     ReduceReads(file_list)
237
238     ### Run UnifiedGenotyper
239     ReUnifiedGenotyper(file_list)
240
241
242     ### BuildErrorModelWithVQSR
243     #BuildErrorModelWithVQSR('snps.raw.vcf', 'SNP')
244     #BuildErrorModelWithVQSR('snps.raw.vcf', 'INDEL')
245
246     ### close the logfile
247     o.close()

```

Listing 2: vcfParser

```

1 #-----+
2 #
3 # 119_vcfParser.py - script to parse vcf format file |
4 #
5 #-----+
6 #
7 # AUTHOR: Vikas Gupta |
8 # CONTACT: vikas0633@gmail.com |
9 # STARTED: 09/06/2013 |
10 # UPDATED: 09/06/2013 |
11 #
12 # DESCRIPTION: |
13 # Short script to convert and copy the wheat BACs |
14 # Run this in the parent dir that the HEX* dirs exist |
15 #
16 # LICENSE: |
17 # GNU General Public License, Version 3 |
18 # http://www.gnu.org/licenses/gpl.html |
19 #
20 #-----+

```

```

21
22 # Example:
23 # python ~/Desktop/script/python/119_vcfParser.py -i.snp.90.PhredQual_5000.vcf
24
25
26 ##### import modules
27 import os,sys,getopt, re, classVCF
28
29
30 ##### global variables
31 global ifile, HEADER
32
33 ##### make a logfile
34 import datetime
35 now = datetime.datetime.now()
36 o = open(str(now.strftime("%Y-%m-%d_%H%M"))+'logfile','w')
37
38
39
40 ##### write logfile
41
42 def logfile(ifile):
43     o.write("Program used: \t\t%s" % "100b_fasta2flat.py"+'\n')
44     o.write("Program was run at: \t%s" % str(now.strftime("%Y-%m-%d_%H%M"))+'\n')
45     o.write("Infile used: \t\t%s" % ifile+'\n')
46
47
48 def help():
49     print '''
50         python 100b_fasta2flat.py -i <infile>
51         '''
52     sys.exit(2)
53
54 ##### main argument to
55
56 def options(argv):
57     global ifile
58     ifile = ''
59     try:
60         opts, args = getopt.getopt(argv,"hi:",["ifile"])
61     except getopt.GetoptError:
62         help()
63     for opt, arg in opts:
64         if opt == '-h':
65             help()
66         elif opt in ("-i", "--ifile"):
67             ifile = arg
68
69     logfile(ifile)
70
71 ##### check if file empty
72 def empty_file(ifile):
73     if os.stat(ifile).st_size==0:
74         sys.exit('File is empty')
75
76
77 def parseFile(ifile):
78     o = open(ifile+'.MG20filtered','w')
79     global HEADER

```

```

80     count = 0
81     for line in open(ifile,'r'):
82         if len(line) > 1 and not line.startswith('##'):
83             line = line.strip('\n')
84             if line.startswith('#CHROM'):
85                 o.write(line+'\n')
86             HEADER = line
87             samples_het = []
88             samples_homo = []
89             sample_names = line.split('\t')[9:]
90             samples_len = len(line.split('\t')) - 9
91             for i in range(samples_len):
92                 samples_het.append(0)
93                 samples_homo.append(0)
94         else:
95             obj = classVCF.VCF(line)
96             genotypes = obj.genotypes()
97             ### check if the MG20 is 0/0 reference Homozygous
98             if obj.genotype(2) == '0/0':
99                 o.write(line+'\n')
100            count += 1
101            genotypes = obj.genotypes()
102            for i in range(len(genotypes)):
103                if obj.genotype(i) == '0/1' or obj.genotype(i) == '1/0':
104                    samples_het[i] += 1
105                elif obj.genotype(i) == '0/0' or obj.genotype(i) == '1/1':
106                    samples_homo[i] += 1
107            print 'Marksers used: ',count
108            print 'Sample\tHetCount\tHomoCount\tHetPer\tHomoPer'
109
110            for i in range(len(sample_names)):
111                total = int(samples_het[i]) + int(samples_homo[i])
112                Het_per = float(samples_het[i])/total
113                Homo_per = float(samples_homo[i])/total
114                print sample_names[i] + '\t' + str(samples_het[i]) + '\t' + str(
115                    samples_homo[i]) + '\t' + str(Het_per) + '\t' + str(Homo_per)
116            o.close()
117
118    if __name__ == "__main__":
119        options(sys.argv[1:])
120        empty_file(ifile)
121
122        parseFile(ifile)
123
124
125        ### close the logfile
126        o.close()

```

Listing 3: classVCF

```

1
2
3 import re
4
5 ### split line
6 def split_line(line):
7     return line.strip().split('\t')
8
9 class VCF:

```

```

10     def __init__(self, line):
11
12         tokens = split_line(line)
13         self.CHROM = tokens[0]
14         self.POS = tokens[1]
15         self.ID = tokens[2]
16         self.REF = tokens[3]
17         self.ALT = tokens[4]
18         self.QUAL = tokens[5]
19         self.FILTER = tokens[6]
20         self.INFO = tokens[7]
21         self.FORMAT = tokens[8]
22
23         self.GENOTYPE = []
24
25         for i in tokens[9:]:
26             self.GENOTYPE.append(i)
27
28
29     def __str__(self):
30         return self.CHROM+'\t'+self.POS
31
32     def chroms(self):
33         return self.CHROM
34
35     def poss(self):
36         return self.POS
37
38     def ids(self):
39         return self.ID
40
41     def refs(self):
42         return self.REF
43
44     def alts(self):
45         return self.ALT
46
47     def quals(self):
48         return self.QUAL
49
50     def filters(self):
51         return self.FILTER
52
53     def infos(self):
54         return self.INFO
55
56     def formats(self):
57         return self.FORMAT
58
59     def genotypes(self):
60         return self.GENOTYPE
61
62     def depth(self):
63         match = re.search(r'DP=.+;', self.INFO)
64         if match:
65             return match.group().split(';')[0].replace('DP=', '')
66         else:
67             return 0
68

```

```

69     def genotype(self, i):
70         if len(self.GENOTYPE[i].split(':')) > 1:
71             return self.GENOTYPE[i].split(':')[0]
72         else:
73             return 'NONE'
74
75     def genotypeDepth(self, i):
76         if len(self.GENOTYPE[i].split(':')) > 1:
77             if len(self.GENOTYPE[i].split(':')) > self.FORMAT.split(':').index('DP')
78                 and self.genotype(i) != './.':
79                 return self.GENOTYPE[i].split(':')[self.FORMAT.split(':').index('DP')
80                                         :]
81             else:
82                 return 'NONE'
83         else:
84             return 0
85
86     def genotypeQual(self, i):
87         if len(self.GENOTYPE[i].split(':')) > 1:
88             if len(self.GENOTYPE[i].split(':')) > self.FORMAT.split(':').index('GQ')
89                 and self.genotype(i) != './.':
90                 return self.GENOTYPE[i].split(':')[self.FORMAT.split(':').index('GQ')
91                                         :]
92             else:
93                 return 0
94         else:
95             return 0
96
97     def genotypeDepthSUM(self):
98         geno_sum = 0
99         for i in self.GENOTYPE:
100            if len(i.split(':')) > 1:
101                if len(self.GENOTYPE[i].split(':')) > self.FORMAT.split(':').index(
102                    'DP'):
103                    geno_sum += int(i.split(':')[self.FORMAT.split(':').index('DP')
104                                     :])
105
106        return geno_sum
107
108    def genotypeCalls(self):
109        geno_call = 0
110        for i in self.GENOTYPE:
111            if len(i.split(':')) > 1:
112                geno_call += 1
113
114        return geno_call
115
116    def genotypeCallsHete(self):
117        geno_call_hete = 0
118        for i in self.GENOTYPE:
119            if len(i.split(':')) > 1:
120                if i.split(':')[0] == '0/1' or i.split(':')[0] == '1/0':
121                    geno_call_hete += 1
122
123        return geno_call_hete
124
125    def genotypeCallsHomo(self):
126        geno_call_homo = 0
127        for i in self.GENOTYPE:
128            if len(i.split(':')) > 1:
129                if i.split(':')[0] == '0/0' or i.split(':')[0] == '1/1':
130                    geno_call_homo += 1

```

```
122     return geno_call_homo
123
124     def InbreedingCoeffs(self):
125         match = re.search(r'InbreedingCoeff=.+;',self.INFO)
126         if match:
127             return match.group().split(';')[0].replace('InbreedingCoeff=', '')
128         else:
129             return 0
130
131     def HaplotypeScores(self):
132         match = re.search(r'HaplotypeScore=.+;',self.INFO)
133         if match:
134             return match.group().split(';')[0].replace('HaplotypeScore=', '')
135         else:
136             return 0
137
138     def variants(self):
139         if self.ALT == '.':
140             return 0
141         else:
142             return 1
```