基因组重测序分析培训

学习文档

目录

一、参考基因组比对

BWA (Burrows-Wheeler Aligner)是一个用于 DNA 序列比对的快速、高效、可扩展的软件工具。

1.1 bwa 软件安装

(1) 源码安装

从官方网站(https://sourceforge.net/projects/bio-bwa/files/)下载最新版本的 bwa 压缩包:

```
wget https://sourceforge.net/projects/bio-bwa/files/bwa-0.7.17.tar.bz2
```

解压缩压缩包并进入 bwa 目录。

```
tar -jxf bwa-0.7.17.tar.bz2
cd bwa-0.7.17
```

编译并添加环境变量

```
make
echo 'PATH=$PATH:/your/bwa/path' >> ~/.bashrc
source ~/.bashrc
```

(2) conda 安装

```
conda create --name bwa -y
conda activate bwa
conda install -c bioconda bwa -y
```

1.2 bwa 软件使用

1.2.1 构建参考基因组索引

bwa index [–p prefix] [–a algoType] <in.db.fasta>

常用选项：

-p :指定输出文件前缀，默认和输入文件名一致，且在输入文件所在的目录；

-a, [is|bwtsw]　构建 index 的算法；

有两个算法：is 是默认的算法，虽然相对较快，但是需要较大的内存，当构建的数据库大于 2GB 的时候就不能正常工作了。 bwtsw 对于短的参考序列式不工作的，必须要大于等于 10MB，但能用于较大的基因组数据，比如人的基因组。

示例：bwa index ref/genome.fa

1.2.2 比对参考基因组

BWA–MEM 算法可进行局部比对。且比对结果可以与 Picard 的 markDuplicates 程序兼容。

用法：

bwa mem [options] ref.fa reads.R1.fq reads.R2.fq

-t INT　线程数，默认是 1。

-M　将 shorter split hits 标记为次优，以兼容 Picard’s markDuplicates 软件。

-p 若无此参数：输入文件只有 1 个，则进行单端比对；若输入文件有 2 个，则作为 paired reads 进行比对。若加入此参数：则仅以第 1 个文件作为输入(输入的文件若有 2 个，则忽略之)，该文件必须是 read1.fq 和 read2.fa 进行 reads 交叉的数据。

-R STR 完整的 read group 的头部，可以用 '\t' 作为分隔符， 在输出的 SAM 文件中被解释为制表符 TAB. read group 的 ID，会被添加到输出文件的每一个 read 的头部。

-T INT 当比对的分值比 INT 小时，不输出该比对结果，这个参数只影响输出的结果，不影响比对的过程。

-a 将所有的比对结果都输出，包括 single-end 和 unpaired paired-end 的 reads，但是这些比对的结果会被标记为次优。

示例：bwa mem -M -t 4 -R "@RG\\tID:1\\tSM:sampleA\\tPL:Illumina\\tLB:lib1\\tPU:unit1" ref/genome.fa Clean_data/sampleA.R1.fq.gz Clean_data/sampleA.R2.fq.gz > result/1.bwa_map/sampleA.bwa.sam

1.3 比对结果处理

Samtools 是一个用来处理 SAM/BAM 格式的比对文件的工具，它能够输入和输出 SAM/BAM 格式的文件，对其进行排序、合并、建立索引等处理。排序后的 BAM 文件通常用于后续转录本的组装与定量。

对比对结果的 sam 文件进行排序并转换成 bam 文件：

示例：samtools sort -@ 8 -o result/1.bwa_map/sampleA.bwa.sorted.bam result/1.bwa_map/sampleA.bwa.sam

可选参数说明：

sort　排序命令

-@　线程数，默认是单线程；

-o　设置最终排序后的输出文件名；

二、变异检测

GATK 是一个应用于前沿科学研究的软件，不断在更新和修正，因此，在使用 GATK 进行变异检测时，最好是下载最新的版本。

2.1 GATK 软件安装

GATK 官方网站：software.broadinstitute.org

gatk4 参考文件下载：console.cloud.google.com

conda 安装

```
conda install -c bioconda gatk -y
```

2.2 GATK 软件使用

2.2.1 标记重复—MarkDuplicates

在数据预处理中，有一个很重要的步骤就是标记重复序列。其中 GATK 的 MarkDuplicates 模块就是专门用来标记重复序列的。

用法：

gatk \

--java-options "-Xmx20G -XX:ParallelGCThreads=8" \ # 指定内存和线程数

MarkDuplicates \ # 使用的 gatk 模块

--REMOVE_DUPLICATES false # 是否删除重复序列，如果是 false，表示只标记重复序列，不删除

-I input.bam \ # 指定输入的 bam 文件（排过序的）

-M metrc.csv \ # 会输出一个 metrics 文件用来记录单端和双端 reads 的重复数目

-O marked.bam # 指定输出的 bam 文件

对 bam 文件构建索引：

samtools index -@ 4 marked.bam

实操示例：gatk --java-options "-Xmx20G -XX:ParallelGCThreads=8" MarkDuplicates --REMOVE_DUPLICATES false -I result/1.bwa_map/sampleA.bwa.sorted.bam -O

result/2.gatk/mark_dup/sampleA.bwa.sorted.mkdup.bam -M result/2.gatk/mark_dup/sampleA.bwa.sorted.mkdup_metrics.txt

samtools index -@ 4 result/2.gatk/mark_dup/sampleA.bwa.sorted.mkdup.bam

2.2.2 构建参考基因组索引文件

（1）用 CreateSequenceDictionary 模块构建.Dict 索引文件：

gatk CreateSequenceDictionary -R ref/genome.fa

（2）用 samtools 构建.fai 索引文件：

samtools faidx ref/genome.fa

2.2.3 生成 gvcf 文件—HaplotypeCaller

用法：

```
gatk \
--java-options "-Xmx20G -XX:ParallelGCThreads=8" \   # 指定内存和线程数
HaplotypeCaller \          # 使用的 gatk 模块
-R genome.fa               # 指定参考基因组（提前建好.dict 和.fai 索引）
-I input.bam \             # 指定输入的 bam 文件（标记过重复的）
-ERC GVCF \                # 表示生成 gvcf 文件以支持下游多样本联合分析
    #1. NONE：只记录变异位点
    #2. BP_RESOLUTION：记录变异和非变异位点，每个位点信息都展示
```

#3. GVCF：记录变异和非变异位点，其中非变异位点已 block 块展示

-L chr1.bed # 指定变异检测的区间，可以是 bed 文件也可以是染色体名字

-O sample.gvcf.gz # 指定输出的 gvcf 文件名

实操：gatk --java-options "-Xmx20G -XX:ParallelGCThreads=8" HaplotypeCaller -R ref/genome.fa -I result/2.gatk/mark_dup/sampleA.bwa.sorted.mkdup.bam -ERC GVCF -O result/2.gatk/HaplotypeCaller_gvcf/sampleA.gvcf.gz

2.2.4 多样本 gvcf 文件合并—CombineGVCFs

用法：

gatk \

--java-options "-Xmx20G -XX:ParallelGCThreads=8" \ # 指定内存和线程数

CombineGVCFs \ # 使用的 gatk 模块

-R genome.fa # 指定参考基因组（提前建好.dict 和.fai 索引）

--variant sample1.gvcf.gz \

--variant sample2.gvcf.gz \

--variant sample3.gvcf.gz \ # 指定要合并的每个样本的 gvcf 文件

-O output.gvcf.gz # 指定合并后的输出文件名

实操：gatk --java-options "-Xmx20G -XX:ParallelGCThreads=8" CombineGVCFs -R ref/genome.fa --variant result/2.gatk/HaplotypeCaller_gvcf/sampleA.gvcf.gz --variant result/2.gatk/HaplotypeCaller_gvcf/sampleB.gvcf.gz --variant result/2.gatk/HaplotypeCaller_gvcf/sampleC.gvcf.gz -O result/2.gatk/Combinegvcf/all.gvcf.gz

2.2.5 获取具体基因型，完成变异检测—GenotypeGVCFs

用法：

```
gatk \
--java-options "-Xmx20G -XX:ParallelGCThreads=8" \   # 指定内存和线程数
GenotypeGVCFs \          # 使用的 gatk 模块
-R genome.fa             # 指定参考基因组（提前建好.dict 和.fai 索引）
-V chr1.gvcf.gz \
-V chr2.gvcf.gz \
-V chr3.gvcf.gz \          # 指定合并所有样本的 gvcf 文件
-O output.vcf.gz           # 指定合并后的输出文件名
```

实操：gatk --java-options "-Xmx20G -XX:ParallelGCThreads=8" GenotypeGVCFs -R ref/genome.fa -V

result/2.gatk/Combinegvcf/all.gvcf.gz -O result/2.gatk/genotypegvcf/raw.combine.vcf.gz

2.2.6 变异位点质控—SelectVariants

拆分 snp 和 indel 两种变异类型

用法：

```
gatk \
--java-options "-Xmx20G -XX:ParallelGCThreads=8" \
SelectVariants \          # 使用的 gatk 模块
-R genome.fa              # 指定参考基因组（提前建好.dict 和.fai 索引）
-V input.vcf.gz \         # 输入文件
-select-type SNP \        # 指定要提取的变异类型【SNP/INDEL】
-O snp.vcf.gz             # 提取指定变异的输出文件名
```

实操：提取 snp

```
gatk --java-options "-Xmx20G -XX:ParallelGCThreads=8" SelectVariants -R ref/genome.fa -V result/2.gatk/genotypegvcf/raw.combine.vcf.gz -select-type SNP -O result/2.gatk/selectVariants/snp.combine.vcf.gz
```

提取 indel：

```
gatk --java-options "-Xmx20G -XX:ParallelGCThreads=8" SelectVariants -R ref/genome.fa -V
```

result/2.gatk/genotypegvcf/raw.combine.vcf.gz -select-type INDEL -O result/2.gatk/selectVariants/indel.combine.vcf.gz

2.2.7 变异位点质控—VariantFiltration

自行设置过滤标准进行硬过滤

用法：

```
gatk \
--java-options "-Xmx20G -XX:ParallelGCThreads=8" \
SelectVariants \          # 使用的 gatk 模块
-R genome.fa              # 指定参考基因组（提前建好.dict 和.fai 索引）
--filter-expression "QD < 10.0" \   # 设置过滤标准
--filter-name Filter \      # 设置过滤掉的位点标记的名字
-V input.vcf.gz \           # 输入文件
-O filter.vcf.gz            # 指定过滤后的输出文件名
```

gatk 官网推荐硬过滤标准参考：

snp：

QUAL < 40 || QD < 2.0 || FS > 60.0 || MQ < 40.0 || SOR > 3.0 || MQRankSum < -12.5 || ReadPosRankSum < -8.0

Indel：

QUAL < 40 || QD < 2.0 || FS > 200.0 || MQ < 40.0 || SOR > 10.0 || MQRankSum < -12.5 || ReadPosRankSum < -20.0

实操：过滤 snp

```
gatk --java-options "-Xmx20G -XX:ParallelGCThreads=8" VariantFiltration -R ref/genome.fa --filter-expression "QUAL < 40 || QD < 2.0 || FS > 60.0 || MQ < 40.0 || SOR > 3.0 || MQRankSum < -12.5 || ReadPosRankSum < -8.0" --filter-name Filter -V result/2.gatk/selectVariants/snp.combine.vcf.gz -O result/2.gatk/filter/snp.filter.vcf.gz
```

过滤 indel：

```
gatk --java-options "-Xmx20G -XX:ParallelGCThreads=8" VariantFiltration -R ref/genome.fa --filter-expression "QUAL < 40 || QD < 2.0 || FS > 200.0 || MQ < 40.0 || SOR > 10.0 || MQRankSum < -12.5 || ReadPosRankSum < -20.0" --filter-name Filter -V result/2.gatk/selectVariants/indel.combine.vcf.gz -O result/2.gatk/filter/indel.filter.vcf.gz
```

2.2.8 变异位点质控—bcftools

提取通过过滤标准的变异位点

```
bcftools filter \                    # 软件程序
-i 'FILTER="PASS"' \                 # 过滤字段
-O z \                               # 输出文件格式（z:gz 格式；v:vcf 格式）
-o snp.filter.final.vcf.gz \         # 过滤后的输出文件名
snp.filter.vcf.gz                    # 过滤前的输入文件名
```

实操：

snp：

bcftools filter -i 'FILTER="PASS"' -O z -o result/2.gatk/filter/snp.filter.final.vcf.gz result/2.gatk/filter/snp.filter.vcf.gz

indel：

bcftools filter -i 'FILTER="PASS"' -O z -o result/2.gatk/filter/indel.filter.final.vcf.gz result/2.gatk/filter/indel.filter.vcf.gz

三、变异位点注释

3.1 snpEff 软件安装

（1）软件下载

wget http://sourceforge.net/projects/snpeff/files/snpEff_latest_core.zip

（2）解压

unzip snpEff_latest_core.zip

3.2 snpEff 软件使用

3.2.1 查询可用数据库

运行命令：java -jar snpEff.jar databases > snpEff.databases

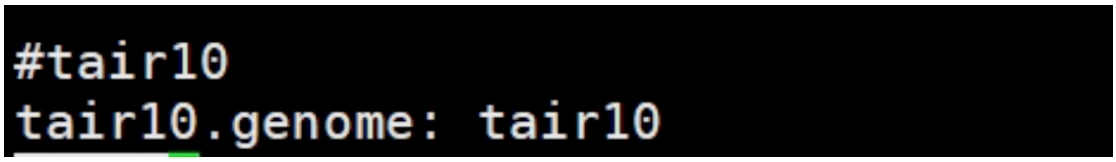
在 snpEff.databases 中，42791 个数据库可供下载，并列出了下载的基因组名、物种及对应的链接。

3.2.2 下载数据库（以大麦为例）

java –jar snpEff.jar download Hordeum_vulgare

3.2.3 配置自己的基因组和注释文件

打开 snpEFF 文件夹下的 snpEff.contig，在 Third party databases 下面增加新的物种信息（以拟南芥 tair10 基因组为例）




```
# 在 SnpEff 目录下，创建目录 data, data/tair10
cd snpEff
mkdir –p data/tair10

# 将 gff3 (.gff) 文件放入 data/tair10 目录下，并改名为 genes.gff
# 将基因组序列文件（.fasta）放入 data/tair10 目录下，并改名为 sequences.fa
# 构建基因组数据库的命令如下（注：在 snpEff 目录下，执行命令）：
java –jar snpEff.jar build –gff3 –v tair10 –noCheckCds –noCheckProtein

# –noCheckCds 不检查 CDS 区
# –noCheckProtein 不检查蛋白质
#如果不加上述两个参数，snpEff 软件默认检查，则需要我们提供物种相应的 CDS 及蛋白质的 fa 文件
```

3.2.3 变异位点注释分析

输入数据：vcf 格式的 snp 文件和 indel 文件

java -Xmx10G -jar snpEff.jar \

-v \　　# 指定输入文件为 vcf 格式

-ud 2000 \　# 指定注释为基因的上下游区域的范围（2kb）

-stats snp.eff.summary.html \　# 注释统计结果

tair10 \　　# 基因组数据库名称

snp.filter.vcf.gz \　# vcf 格式的注释输入文件

>snp.eff.anno.vcf　　# vcf 格式的注释输出文件

实操命令：

注释 snp：

java -Xmx10G -jar snpEff.jar –v -ud 2000 -stats snp.eff.summary.html

tair10 snp.filter.vcf.gz > snp.eff.anno.vcf 2>snpeff.snp.log

注释 indel：

java -Xmx10G -jar snpEff.jar –v -ud 2000 -stats indel.eff.summary.html

tair10 indel.filter.vcf.gz > indel.eff.anno.vcf 2>snpeff.indel.log

3.2.4 注释结果解读

结果有两个文件：snp.eff.anno.vcf 和 snp.eff.summary.html

snp.eff.anno.vcf 文件中，注释信息以“|”分割：

1: Allele：T 表示该突变碱基的类型。

2: Annotation：突变类型（3_prime_UTR_variant,

5_prime_UTR_premature_start_codon_gain_variant, 5_prime_UTR_variant,

downstream_gene_variant, initiator_codo_variant, intergenic_region,

intron_variant, missense_variant, non_canonical_start_codon,

non_coding_transcript_exon_variant, splice_acceptor_variant, splice_donor_variant, splice_region_variant, start_lost, stop_gained, stop_lost, stop_retained_variant, synonymous_variant, upstream_gene_variant） 多个类型之间用&连接。

3: Annotation_impact：对变异位点产生的影响程度进行简单评估，有四个程度（HIGH, MODERATE, LOW, MODIFIER）。

4: Gene_Name：该变异位点所在基因的基因名，如果变异位点的突变类型是 intergenic_region，则显示的是离该变异位点最近的一个基因。

5: Gene_ID：基因 ID。

6: Feature_Type：变异位点所在的区域类型，transcript，motif，miRNA。

7: Feature_ID：Feature_Type 所对应的 ID。

8： Transcript_BioType：转录本类型。

9: Rank：只有当变异位点位于基因区域时才有值，当变异位点位于基因区域以外（intergenic_region）时，该字段的值为空。该值给出的是变异位点所处的 exton/intron 和该基因的 exon/intron 的总数。

10: HGVS.c：在 DNA 水平上，采用 HGVS 标准命名的变异位点的情况。

11: HVGS.p：在蛋白质水平上，采用 HGVS 标准命名的变异位点的情况。

12: cDNA.pos/cDNA.length：变异位点在 cDNA 上的位置/cDNA 的长度。

13: CDS.pos/CDS.length：变异位点在 CDS 的位置/CDS 的长度。

14: AA.pos/AA.length：变异位点在氨基酸上的位置/氨基酸的长度。

15: Distance：不同的情况，距离的含义是不同的，因此可能会是空值。

Up/Downstream：到第一个/最后一个密码子的距离。Intergenic：到最近基因的

距离。到外显子中最近的内含子边界的距离（+/-代表上游/下游）。如果相同，使用正数。在内含子中离最近外显子边界的距离（+/-上/下）。到基序中第一碱基的距离。到 miRNA 中第一碱基的距离。在剪接位点或剪接区域中，离外显子-内含子边界的距离 ChipSeq peak：到顶点（或峰中心）的距离。Histone/Histone state：到顶点（或峰中心）的距离。

snp.eff.summary.html 文件在网页中查看

该文件对以下几个方面进行的统计信息，可以根据需要绘制饼图或者柱形图。

详细结果解读见 PPT

四、结构变异检测

输入文件：标记重复后的 bam 文件

软件：samtools、lumpy

4.1 lumpy 检测 sv 分析

第一步：提取非正常比对的 reads

```
samtools view \          # 软件程序
-b \                     # 指定输出文件为 bam 格式
-F 1294 \                # flag 值代表的含义可在该网站查询
（https://broadinstitute.github.io/picard/explain-flags.html）
-@ 2 \                   # 指定线程数
sampleA.bwa.sorted.mkdup.bam \     # 样本的 bam 文件
```

> sampleA.discordants.bam　　# 提取的非正常比对的 reads

实操：samtools view -b -F 1294 -@ 2 result/2.gatk/mark_dup/sampleA.bwa.sorted.mkdup.bam > result/4.sv/sampleA.discordants.bam

第二步：提取分裂比对的 reads

① 将 bam 格式文件转换成 sam 格式：

samtools view \　　# 软件程序

-h \　　# 指定输出的 sam 文件带 header 信息(默认是不带的)

sampleA.bwa.sorted.mkdup.bam \　　# 样本的 bam 文件

> sampleA.output.sam　　# 输出的 sam 文件名

② 提取分裂比对的 reads：

extractSplitReads_BwaMem \　　# lumpy 软件的子程序

-i sampleA.output.sam \　　# 样本的 sam 文件

> sampleA.split.sam　　# 提取的分裂比对的 reads（ sam 格式 ）

③ 将 sam 格式转换成 bam 格式：

samtools view \　　# 软件程序

-Sb \　　# 指定输入文件为 sam 格式 ,同时输出文件为 bam 格式

sampleA.split.sam \　　# 样本分裂比对的 sam 文件

> sampleA.split.bam　　# 输出的样本分裂比对的 bam 文件名

实操：

samtools view -h
result/2.gatk/mark_dup/sampleA.bwa.sorted.mkdup.bam |
extractSplitReads_BwaMem -i stdin | samtools view -Sb - >
result/4.sv/sampleA.splitters.bam

第三步：结构变异 SV 检测

lumpyexpress \ # 软件名称
-B sampleA.bwa.sorted.mkdup.bam \ #指定样本的 bam 文件(标记重复)
-S sampleA.split.bam \ # 样本分裂比对的 bam 文件
-D sampleA.discordants.bam \ # 样本非正常比对的 bam 文件
-o sampleA.vcf # 结构变异检测结果文件

实操：

lumpyexpress -B
result/2.gatk/mark_dup/sampleA.bwa.sorted.mkdup.bam -S
result/4.sv/sampleA.splitters.bam -D
result/4.sv/sampleA.discordants.bam -o result/4.sv/sampleA.vcf