GenomicRanges HOWTOs

Bioconductor Team

Edited: January 2018; Compiled: October 27, 2020

Contents

1 Introduction

1.1 Purpose of this document

This document is a collection of *HOWTOs*. Each *HOWTO* is a short section that demonstrates how to use the containers and operations implemented in the *GenomicRanges* and related packages (*IRanges*, *Biostrings*, *Rsamtools*, *GenomicAlignments*, *BSgenome*, and *GenomicFeatures*) to perform a task typically found in the context of a high throughput sequence analysis.

Unless stated otherwise, the *HOWTOs* are self contained, independent of each other, and can be studied and reproduced in any order.

1.2 Prerequisites and additional recommended reading

We assume the reader has some previous experience with *R* and with basic manipulation of `GRanges`, `GRangesList`, `Rle`, `RleList`, and `DataFrame` objects. See the "An Introduction to Genomic Ranges Classes" vignette located in the *GenomicRanges* package (in the same folder as this document) for an introduction to these containers.

Additional recommended readings after this document are the "Software for Computing and Annotating Genomic Ranges" paper[Lawrence et al. (2013)] and the "Counting reads with `summarizeOverlaps`" vignette located in the *GenomicAlignments* package.

To display the list of vignettes available in the *GenomicRanges* package, use `browseVignettes("GenomicRanges")`.

1.3 Input data and terminology used across the HOWTOs

In order to avoid repetition, input data, concepts and terms used in more than one *HOWTO* are described here:

- **The *pasillaBamSubset* data package**: contains both a BAM file with single-end reads (untreated1_chr4) and a BAM file with paired-end reads (untreated3_chr4). Each file is a subset of chr4 from the "Pasilla" experiment.

```
> library(pasillaBamSubset)
> untreated1_chr4()

[1] "/home/biocbuild/bbs-3.12-bioc/R/library/pasillaBamSubset/extdata/untreated1_chr4.bam"

> untreated3_chr4()

[1] "/home/biocbuild/bbs-3.12-bioc/R/library/pasillaBamSubset/extdata/untreated3_chr4.bam"
```

  See `?pasillaBamSubset` for more information.

```
> ?pasillaBamSubset
```

- **Gene models and *TxDb* objects**: A *gene model* is essentially a set of annotations that describes the genomic locations of the known genes, transcripts, exons, and CDS, for a given organism. In *Bioconductor* it is typically represented as a *TxDb* object but also sometimes as a *GRanges* or *GRangesList* object. The *GenomicFeatures* package contains tools for making and manipulating *TxDb* objects.

2 HOWTOs

2.1 How to read single-end reads from a BAM file

As sample data we use the *pasillaBamSubset* data package described in the introduction.

```
> library(pasillaBamSubset)
> un1 <- untreated1_chr4()  # single-end reads
```

Several functions are available for reading BAM files into *R*:

```
readGAlignments()
readGAlignmentPairs()
readGAlignmentsList()
scanBam()
```

`scanBam` is a low-level function that returns a list of lists and is not discussed further here. See `?scanBam` in the *Rsamtools* package for more information.

Single-end reads can be loaded with the `readGAlignments` function from the *GenomicAlignments* package.

```
> library(GenomicAlignments)
> gal <- readGAlignments(un1)
```

Data subsets can be specified by genomic position, field names, or flag criteria in the `ScanBamParam`. Here we input records that overlap position 1 to 5000 on the negative strand with `flag` and `cigar` as metadata columns.

```
> what <- c("flag", "cigar")
> which <- GRanges("chr4", IRanges(1, 5000))
> flag <- scanBamFlag(isMinusStrand = TRUE)
> param <- ScanBamParam(which=which, what=what, flag=flag)
> neg <- readGAlignments(un1, param=param)
> neg

GAlignments object with 37 alignments and 2 metadata columns:
       seqnames strand       cigar    qwidth     start       end
          <Rle>  <Rle> <character> <integer> <integer> <integer>
   [1]     chr4      -         75M        75       892       966
   [2]     chr4      -         75M        75       919       993
   [3]     chr4      -         75M        75       967      1041
   ...      ...    ...         ...       ...       ...       ...
```

```
  [35]     chr4      -         75M        75      4997      5071
  [36]     chr4      -         75M        75      4998      5072
  [37]     chr4      -         75M        75      4999      5073
           width     njunc |      flag       cigar
       <integer> <integer> | <integer> <character>
   [1]        75         0 |        16         75M
   [2]        75         0 |        16         75M
   [3]        75         0 |        16         75M
   ...       ...       ... .       ...         ...
  [35]        75         0 |        16         75M
  [36]        75         0 |        16         75M
  [37]        75         0 |        16         75M
  -------
  seqinfo: 8 sequences from an unspecified genome
```

Another approach to subsetting the data is to use `filterBam`. This function creates a new BAM file of records passing user-defined criteria. See `?filterBam` in the *Rsamtools* package for more information.

2.2 How to read paired-end reads from a BAM file

As sample data we use the *pasillaBamSubset* data package described in the introduction.

```
> library(pasillaBamSubset)
> un3 <- untreated3_chr4()  # paired-end reads
```

Paired-end reads can be loaded with the `readGAlignmentPairs` or `readGAlignmentsList` function from the *GenomicAlignments* package. These functions use the same mate paring algorithm but output different objects.

Let's start with `readGAlignmentPairs`:

```
> un3 <- untreated3_chr4()
> gapairs <- readGAlignmentPairs(un3)
```

The `GAlignmentPairs` class holds only pairs; reads with no mate or with ambiguous pairing are discarded. Each list element holds exactly 2 records (a mated pair). Records can be accessed as the `first` and`last` segments in a template or as `left` and `right` alignments. See `?GAlignmentPairs` in the *GenomicAlignments* package for more information.

```
> gapairs

GAlignmentPairs object with 75409 pairs, strandMode=1, and 0 metadata columns:
          seqnames strand   :           ranges  --           ranges
             <Rle>  <Rle>   :        <IRanges>  --        <IRanges>
      [1]     chr4      +   :          169-205  --          326-362
      [2]     chr4      +   :          943-979  --        1086-1122
      [3]     chr4      +   :          944-980  --        1119-1155
      ...      ...    ... ...              ... ...              ...
  [75407]     chr4      +   : 1348217-1348253  -- 1348215-1348251
  [75408]     chr4      +   : 1349196-1349232  -- 1349326-1349362
```

```
  [75409]     chr4      +   : 1349708-1349744  -- 1349838-1349874
  -------
  seqinfo: 8 sequences from an unspecified genome
```

For `readGAlignmentsList`, mate pairing is performed when `asMates` is set to `TRUE` on the `BamFile` object, otherwise records are treated as single-end.

```
> galist <- readGAlignmentsList(BamFile(un3, asMates=TRUE))
```

`GAlignmentsList` is a more general 'list-like' structure that holds mate pairs as well as non-mates (i.e., singletons, records with unmapped mates etc.) A `mates_status` metadata column (accessed with `mcols`) indicates which records were paired.

```
> galist

GAlignmentsList object of length 96636:
[[1]]
GAlignments object with 2 alignments and 0 metadata columns:
      seqnames strand       cigar    qwidth     start       end
         <Rle>  <Rle> <character> <integer> <integer> <integer>
  [1]     chr4      +         37M        37       169       205
  [2]     chr4      -         37M        37       326       362
          width     njunc
      <integer> <integer>
  [1]        37         0
  [2]        37         0
  -------
  seqinfo: 8 sequences from an unspecified genome

[[2]]
GAlignments object with 2 alignments and 0 metadata columns:
      seqnames strand       cigar    qwidth     start       end
         <Rle>  <Rle> <character> <integer> <integer> <integer>
  [1]     chr4      +         37M        37       946       982
  [2]     chr4      -         37M        37       986      1022
          width     njunc
      <integer> <integer>
  [1]        37         0
  [2]        37         0
  -------
  seqinfo: 8 sequences from an unspecified genome

[[3]]
GAlignments object with 2 alignments and 0 metadata columns:
      seqnames strand       cigar    qwidth     start       end
         <Rle>  <Rle> <character> <integer> <integer> <integer>
  [1]     chr4      +         37M        37       943       979
  [2]     chr4      -         37M        37      1086      1122
          width     njunc
      <integer> <integer>
  [1]        37         0
  [2]        37         0
```

```
  -------
  seqinfo: 8 sequences from an unspecified genome

...
<96633 more elements>
```

Non-mated reads are returned as groups by QNAME and contain any number of records. Here the non-mate groups range in size from 1 to 9.

```
> non_mates <- galist[unlist(mcols(galist)$mate_status) == "unmated"]
> table(elementNROWS(non_mates))

    1     2     3     4     5     6     7     8     9
18191  2888    69    60     7     8     2     1     1
```

2.3 How to read and process a big BAM file by chunks in order to reduce memory usage

A large BAM file can be iterated through in chunks by setting a `yieldSize` on the *BamFile* object. As sample data we use the *pasillaBamSubset* data package described in the introduction.

```
> library(pasillaBamSubset)
> un1 <- untreated1_chr4()
> bf <- BamFile(un1, yieldSize=100000)
```

Iteration through a BAM file requires that the file be opened, repeatedly queried inside a loop, then closed. Repeated calls to `readGAlignments` without opening the file first result in the same 100000 records returned each time.

```
> open(bf)
> cvg <- NULL
> repeat {
+     chunk <- readGAlignments(bf)
+     if (length(chunk) == 0L)
+         break
+     chunk_cvg <- coverage(chunk)
+     if (is.null(cvg)) {
+         cvg <- chunk_cvg
+     } else {
+         cvg <- cvg + chunk_cvg
+     }
+ }
> close(bf)
> cvg

RleList of length 8
$chr2L
integer-Rle of length 23011544 with 1 run
  Lengths: 23011544
  Values :        0
```

```
$chr2R
integer-Rle of length 21146708 with 1 run
  Lengths: 21146708
  Values :        0

$chr3L
integer-Rle of length 24543557 with 1 run
  Lengths: 24543557
  Values :        0

$chr3R
integer-Rle of length 27905053 with 1 run
  Lengths: 27905053
  Values :        0

$chr4
integer-Rle of length 1351857 with 122061 runs
  Lengths:   891    27     5    12    13    45 ...   106    75 1600    75 1659
  Values :     0     1     2     3     4     5 ...     0     1    0     1    0

...
<3 more elements>
```

2.4 How to compute read coverage

The "read coverage" is the number of reads that cover a given genomic position. Computing the read coverage generally consists in computing the coverage at each position in the genome. This can be done with the `coverage()` function.

As sample data we use the *pasillaBamSubset* data package described in the introduction.

```
> library(pasillaBamSubset)
> un1 <- untreated1_chr4()  # single-end reads
> library(GenomicAlignments)
> reads1 <- readGAlignments(un1)
> cvg1 <- coverage(reads1)
> cvg1

RleList of length 8
$chr2L
integer-Rle of length 23011544 with 1 run
  Lengths: 23011544
  Values :        0

$chr2R
integer-Rle of length 21146708 with 1 run
  Lengths: 21146708
  Values :        0

$chr3L
```

```
integer-Rle of length 24543557 with 1 run
  Lengths: 24543557
  Values :        0

$chr3R
integer-Rle of length 27905053 with 1 run
  Lengths: 27905053
  Values :        0

$chr4
integer-Rle of length 1351857 with 122061 runs
  Lengths:   891    27     5    12    13    45 ...   106    75 1600    75 1659
  Values :     0     1     2     3     4     5 ...     0     1     0     1     0

...
<3 more elements>
```

Coverage on chr4:

```
> cvg1$chr4

integer-Rle of length 1351857 with 122061 runs
  Lengths:   891    27     5    12    13    45 ...   106    75 1600    75 1659
  Values :     0     1     2     3     4     5 ...     0     1     0     1     0
```

Average and max coverage:

```
> mean(cvg1$chr4)

[1] 11.33746

> max(cvg1$chr4)

[1] 5627
```

Note that `coverage()` is a generic function with methods for different types of objects. See `?coverage` for more information.

2.5 How to find peaks in read coverage

ChIP-Seq analysis usually involves finding peaks in read coverage. This process is sometimes called “peak calling” or “peak detection”. Here we’re only showing a naive way to find peaks in the object returned by the `coverage()` function. *Bioconductor* packages *BayesPeak*, *bumphunter*, *Starr*, *CexoR*, *exomePeak*, *RIPSeeker*, and others, provide sophisticated peak calling tools for ChIP-Seq, RIP-Seq, and other kind of high throughput sequencing data.

Let’s assume `cvg1` is the object returned by `coverage()` (see previous *HOWTO* for how to compute it). We can use the `slice()` function to find the genomic regions where the coverage is greater or equal to a given threshold.

```
> chr4_peaks <- slice(cvg1$chr4, lower=500)
> chr4_peaks
```

```
Views on a 1351857-length Rle subject

views:
       start     end width
 [1]   86849   87364   516 [ 525  538  554  580  583  585  589 ...]
 [2]   87466   87810   345 [4924 4928 4941 4943 4972 5026 5039 ...]
 [3]  340791  340798     8 [508 512 506 530 521 519 518 501]
 [4]  340800  340885    86 [500 505 560 560 565 558 564 559 555 ...]
 [5]  348477  348483     7 [503 507 501 524 515 513 512]
 [6]  348488  348571    84 [554 554 559 552 558 553 549 550 559 ...]
 [7]  692512  692530    19 [502 507 508 518 520 522 524 526 547 ...]
 [8]  692551  692657   107 [ 530  549  555  635  645  723  725 ...]
 [9]  692798  692800     3 [503 500 503]
 ...     ...     ...   ... ...
[34] 1054306 1054306     1 [502]
[35] 1054349 1054349     1 [501]
[36] 1054355 1054444    90 [510 521 525 532 532 539 549 555 557 ...]
[37] 1054448 1054476    29 [502 507 516 517 508 517 525 528 532 ...]
[38] 1054479 1054482     4 [504 503 506 507]
[39] 1054509 1054509     1 [500]
[40] 1054511 1054511     1 [502]
[41] 1054521 1054623   103 [529 521 529 530 524 525 547 540 536 ...]
[42] 1054653 1054717    65 [520 519 516 528 526 585 591 589 584 ...]

> length(chr4_peaks)  # nb of peaks

[1] 42
```

The weight of a given peak can be defined as the number of aligned nucleotides that belong to the peak (a.k.a. the area under the peak in mathematics). It can be obtained with `sum()`:

```
> sum(chr4_peaks)

 [1] 1726347 1300700    4115   52301    3575   51233   10382   95103
 [9]    1506     500    2051     500    5834   10382   92163     500
[17]   88678    1512     500   11518   14514    5915    3598    7821
[25]     511     508     503     500    1547    8961   43426   22842
[33]     503     502     501   51881   15116    2020     500     502
[41]   67010   40496
```

2.6 How to retrieve a gene model from the UCSC genome browser

See introduction for a quick description of what *gene models* and *TxDb* objects are. We can use the `makeTranscriptDbFromUCSC()` function from the *GenomicFeatures* package to import a UCSC genome browser track as a *TxDb* object.

```
> library(GenomicFeatures)
> ### Internet connection required! Can take several minutes...
> txdb <- makeTxDbFromUCSC(genome="sacCer2", tablename="ensGene")
```

See `?makeTxDbFromUCSC` in the *GenomicFeatures* package for more information.

Note that some of the most frequently used gene models are available as TxDb packages. A TxDb package consists of a pre-made *TxDb* object wrapped into an annotation data package. Go to http://bioconductor.org/packages/release/BiocViews.html#___TxDb to browse the list of available TxDb packages.

```
> library(TxDb.Hsapiens.UCSC.hg19.knownGene)
> txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
> txdb

TxDb object:
# Db type: TxDb
# Supporting package: GenomicFeatures
# Data source: UCSC
# Genome: hg19
# Organism: Homo sapiens
# Taxonomy ID: 9606
# UCSC Table: knownGene
# Resource URL: http://genome.ucsc.edu/
# Type of Gene ID: Entrez Gene ID
# Full dataset: yes
# miRBase build ID: GRCh37
# transcript_nrow: 82960
# exon_nrow: 289969
# cds_nrow: 237533
# Db created by: GenomicFeatures package from Bioconductor
# Creation time: 2015-10-07 18:11:28 +0000 (Wed, 07 Oct 2015)
# GenomicFeatures version at creation time: 1.21.30
# RSQLite version at creation time: 1.0.0
# DBSCHEMAVERSION: 1.1
```

Extract the transcript coordinates from this gene model:

```
> transcripts(txdb)

GRanges object with 82960 ranges and 2 metadata columns:
                seqnames      ranges strand |     tx_id     tx_name
                   <Rle>   <IRanges>  <Rle> | <integer> <character>
      [1]           chr1 11874-14409      + |         1  uc001aaa.3
      [2]           chr1 11874-14409      + |         2  uc010nxq.1
      [3]           chr1 11874-14409      + |         3  uc010nxr.1
      ...            ...         ...    ... .       ...         ...
  [82958] chrUn_gl000243 11501-11530      + |     82958  uc011mgw.1
  [82959] chrUn_gl000243 13608-13637      + |     82959  uc022brq.1
  [82960] chrUn_gl000247   5787-5816      - |     82960  uc022brr.1
  -------
  seqinfo: 93 sequences (1 circular) from hg19 genome
```

2.7 How to retrieve a gene model from Ensembl

See introduction for a quick description of what *gene models* and *TxDb* objects are. We can use the `makeTranscriptDbFromBiomart()` function from the *GenomicFeatures* package to retrieve a gene model from the Ensembl Mart.

```
> library(GenomicFeatures)
> ### Internet connection required! Can take several minutes...
> txdb <- makeTxDbFromBiomart(biomart="ensembl",
+                             dataset="hsapiens_gene_ensembl")
```

See `?makeTxDbFromBiomart` in the *GenomicFeatures* package for more information.

Note that some of the most frequently used gene models are available as TxDb packages. A TxDb package consists of a pre-made *TxDb* object wrapped into an annotation data package. Go to http://bioconductor.org/packages/release/BiocViews.html#___TxDb to browse the list of available TxDb packages.

```
> library(TxDb.Athaliana.BioMart.plantsmart22)
> txdb <- TxDb.Athaliana.BioMart.plantsmart22
> txdb

TxDb object:
# Db type: TxDb
# Supporting package: GenomicFeatures
# Data source: BioMart
# Organism: Arabidopsis thaliana
# Resource URL: www.biomart.org:80
# BioMart database: plants_mart_22
# BioMart database version: ENSEMBL PLANTS 22 (EBI UK)
# BioMart dataset: athaliana_eg_gene
# BioMart dataset description: Arabidopsis thaliana genes (TAIR10 (2010-09-TAIR10))
# BioMart dataset version: TAIR10 (2010-09-TAIR10)
# Full dataset: yes
# miRBase build ID: NA
# transcript_nrow: 41671
# exon_nrow: 171013
# cds_nrow: 147494
# Db created by: GenomicFeatures package from Bioconductor
# Creation time: 2014-09-26 11:23:54 -0700 (Fri, 26 Sep 2014)
# GenomicFeatures version at creation time: 1.17.17
# RSQLite version at creation time: 0.11.4
# DBSCHEMAVERSION: 1.0
# TaxID: 3702
```

Extract the exon coordinates from this gene model:

```
> exons(txdb)

GRanges object with 171013 ranges and 1 metadata column:
           seqnames         ranges strand |   exon_id
              <Rle>      <IRanges>  <Rle> | <integer>
       [1]        1      3631-3913      + |         1
       [2]        1      3996-4276      + |         2
       [3]        1      4486-4605      + |         3
       ...      ...            ...    ... .       ...
  [171011]       Pt 137869-137940      - |    171011
  [171012]       Pt 144921-145154      - |    171012
  [171013]       Pt 145291-152175      - |    171013
```

```
  -------
  seqinfo: 7 sequences (1 circular) from an unspecified genome
```

2.8 How to load a gene model from a GFF or GTF file

See introduction for a quick description of what *gene models* and *TxDb* objects are. We can use the `makeTranscriptDbFromGFF()` function from the *GenomicFeatures* package to import a GFF or GTF file as a *TxDb* object.

```
> library(GenomicFeatures)
> gff_file <- system.file("extdata", "GFF3_files", "a.gff3",
+                          package="GenomicFeatures")
> txdb <- makeTxDbFromGFF(gff_file, format="gff3")
> txdb

TxDb object:
# Db type: TxDb
# Supporting package: GenomicFeatures
# Data source: /home/biocbuild/bbs-3.12-bioc/R/library/GenomicFeatures/extdata/GFF3_files/a.gff3
# Organism: NA
# Taxonomy ID: NA
# miRBase build ID: NA
# Genome: NA
# Nb of transcripts: 488
# Db created by: GenomicFeatures package from Bioconductor
# Creation time: 2020-10-27 20:24:48 -0400 (Tue, 27 Oct 2020)
# GenomicFeatures version at creation time: 1.42.0
# RSQLite version at creation time: 2.2.1
# DBSCHEMAVERSION: 1.2
```

See `?makeTxDbFromGFF` in the *GenomicFeatures* package for more information.

Extract the exon coordinates grouped by gene from this gene model:

```
> exonsBy(txdb, by="gene")

GRangesList object of length 488:
$Solyc00g005000.2
GRanges object with 2 ranges and 2 metadata columns:
        seqnames      ranges strand |   exon_id            exon_name
           <Rle>   <IRanges>  <Rle> | <integer>          <character>
  [1] SL2.40ch00 16437-17275      + |         1 Solyc00g005000.2.1.1
  [2] SL2.40ch00 17336-18189      + |         2 Solyc00g005000.2.1.2
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths

$Solyc00g005020.1
GRanges object with 3 ranges and 2 metadata columns:
        seqnames      ranges strand |   exon_id            exon_name
           <Rle>   <IRanges>  <Rle> | <integer>          <character>
  [1] SL2.40ch00 68062-68211      + |         3 Solyc00g005020.1.1.1
  [2] SL2.40ch00 68344-68568      + |         4 Solyc00g005020.1.1.2
```

```
  [3] SL2.40ch00 68654-68764      + |         5 Solyc00g005020.1.1.3
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths

$Solyc00g005040.2
GRanges object with 4 ranges and 2 metadata columns:
        seqnames        ranges strand |   exon_id           exon_name
           <Rle>     <IRanges>  <Rle> | <integer>         <character>
  [1] SL2.40ch00 550920-550945      + |         6 Solyc00g005040.2.1.1
  [2] SL2.40ch00 551034-551132      + |         7 Solyc00g005040.2.1.2
  [3] SL2.40ch00 551218-551250      + |         8 Solyc00g005040.2.1.3
  [4] SL2.40ch00 551343-551576      + |         9 Solyc00g005040.2.1.4
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths

...
<485 more elements>
```

2.9 How to retrieve a gene model from *AnnotationHub*

When a gene model is not available as a *GRanges* or *GRangesList* object or as a *Bioconductor* data package, it may be available on *AnnotationHub*. In this *HOWTO*, will look for a gene model for Drosophila melanogaster on *AnnotationHub*. Create a 'hub' and then filter on Drosophila melanogaster:

```
> library(AnnotationHub)
> ### Internet connection required!
> hub <- AnnotationHub()
> hub <- subset(hub, hub$species=='Drosophila melanogaster')
```

There are 87 files that match Drosophila melanogaster. If you look at the metadata in hub, you can see that the 7th record representes a GRanges object from UCSC

```
> length(hub)

[1] 420

> head(names(hub))

[1] "AH6789" "AH6790" "AH6791" "AH6792" "AH6793" "AH6794"

> head(hub$title, n=10)

 [1] "Assembly"       "GDP Insertions" "BAC End Pairs"  "FlyBase Genes"
 [5] "RefSeq Genes"   "Ensembl Genes"  "CONTRAST"       "Human Proteins"
 [9] "Spliced ESTs"   "Other mRNAs"

> ## then look at a specific slice of the hub object.
> hub[7]

AnnotationHub with 1 record
# snapshotDate(): 2020-10-26
# names(): AH6795
```

```
# $dataprovider: UCSC
# $species: Drosophila melanogaster
# $rdataclass: GRanges
# $rdatadateadded: 2013-04-04
# $title: CONTRAST
# $description: GRanges object from on UCSC track ÃćâĆňËIJCONTRASTÃćâĆňâĎć
# $taxonomyid: 7227
# $genome: dm3
# $sourcetype: UCSC track
# $sourceurl: rtracklayer://hgdownload.cse.ucsc.edu/goldenpath/dm3/d...
# $sourcesize: NA
# $tags: c("contrastGene", "UCSC", "track", "Gene",
#   "Transcript", "Annotation")
# retrieve record with 'object[["AH6795"]]'
```

So you can retrieve that dm3 file as a `GRanges` like this:

```
> gr <- hub[[names(hub)[7]]]
> summary(gr)

[1] "GRanges object with 13504 ranges and 5 metadata columns"
```

The metadata fields contain the details of file origin and content.

```
> metadata(gr)

$AnnotationHubName
[1] "AH6795"

$`File Name`
[1] "contrastGene"

$`Data Source`
[1] "rtracklayer://hgdownload.cse.ucsc.edu/goldenpath/dm3/database/contrastGene"

$Provider
[1] "UCSC"

$Organism
[1] "Drosophila melanogaster"

$`Taxonomy ID`
[1] 7227
```

Split the *GRanges* object by gene name to get a *GRangesList* object of transcript ranges grouped by gene.

```
> txbygn <- split(gr, gr$name)
```

You can now use `txbygn` with the `summarizeOverlaps` function to prepare a table of read counts for RNA-Seq differential gene expression.

Note that before passing `txbygn` to `summarizeOverlaps`, you should confirm that the seqlevels (chromosome names) in it match those in the BAM file. See `?renameSeqlevels`, `?keepSeqlevels` and `?seqlevels` for examples of renaming seqlevels.

2.10 How to annotate peaks in read coverage

[coming soon...]

2.11 How to prepare a table of read counts for RNA-Seq differential gene expression

Methods for RNA-Seq gene expression analysis generally require a table of counts that summarize the number of reads that overlap or 'hit' a particular gene. In this *HOWTO* we count with the `summarizeOverlaps` function from the *GenomicAlignments* package and create a count table from the results.

Other packages that provide read counting are *Rsubread* and *easyRNASeq*. The *parathyroidSE* package vignette contains a workflow on counting and other common operations required for differential expression analysis.

As sample data we use the *pasillaBamSubset* data package described in the introduction.

```
> library(pasillaBamSubset)
> reads <- c(untrt1=untreated1_chr4(),  # single-end reads
+            untrt3=untreated3_chr4())  # paired-end reads
```

`summarizeOverlaps` requires the name of a BAM file(s) and a gene model to count against. See introduction for a quick description of what a *gene models* is. The gene model must match the genome build the reads in the BAM file were aligned to. For the pasilla data this is dm3 Dmelanogaster which is available as a *Bioconductor* package. Load the package and extract the exon ranges grouped by gene:

```
> library(TxDb.Dmelanogaster.UCSC.dm3.ensGene)
> exbygene <- exonsBy(TxDb.Dmelanogaster.UCSC.dm3.ensGene, "gene")
```

`exbygene` is a *GRangesList* object with one list element per gene in the gene model.

`summarizeOverlaps` automatically sets a `yieldSize` on large BAM files and iterates over them in chunks. When reading paired-end data set the `singleEnd` argument to FALSE. See ?`summarizeOverlaps` for details reguarding the count `modes` and additional arguments.

```
> library(GenomicAlignments)
> se <- summarizeOverlaps(exbygene, reads, mode="IntersectionNotEmpty")
```

The return object is a `SummarizedExperiment` with counts accessible with the `assays` accessor:

```
> class(se)

[1] "RangedSummarizedExperiment"
attr(,"package")
[1] "SummarizedExperiment"
```

```
> head(table(assays(se)$counts))

    0     1     2     3     4     5 
31188     2     6     3     4     4 
```

The count vector is the same length as `exbygene`:

```
> identical(length(exbygene), length(assays(se)$counts))

[1] FALSE
```

A copy of `exbygene` is stored in the `se` object and accessible with `rowRanges` accessor:

```
> rowRanges(se)

GRangesList object of length 15682:
$FBgn0000003
GRanges object with 1 range and 2 metadata columns:
      seqnames          ranges strand |   exon_id   exon_name
         <Rle>       <IRanges>  <Rle> | <integer> <character>
  [1]    chr3R 2648220-2648518      + |     45123        <NA>
  -------
  seqinfo: 15 sequences (1 circular) from dm3 genome

$FBgn0000008
GRanges object with 13 ranges and 2 metadata columns:
       seqnames            ranges strand |   exon_id   exon_name
          <Rle>         <IRanges>  <Rle> | <integer> <character>
   [1]    chr2R 18024494-18024531      + |     20314        <NA>
   [2]    chr2R 18024496-18024713      + |     20315        <NA>
   [3]    chr2R 18024938-18025756      + |     20316        <NA>
   ...      ...               ...    ... .       ...         ...
  [11]    chr2R 18059821-18059938      + |     20328        <NA>
  [12]    chr2R 18060002-18060339      + |     20329        <NA>
  [13]    chr2R 18060002-18060346      + |     20330        <NA>
  -------
  seqinfo: 15 sequences (1 circular) from dm3 genome

...
<15680 more elements>
```

Two popular packages for RNA-Seq differential gene expression are *DESeq2* and *edgeR*. Tables of counts per gene are required for both and can be easily created with a vector of counts. Here we use the counts from our *SummarizedExperiment* object:

```
> colData(se)$trt <- factor(c("untrt", "untrt"), levels=c("trt", "untrt"))
> colData(se)

DataFrame with 2 rows and 1 column
            trt
       <factor>
untrt1    untrt
untrt3    untrt

> library(DESeq2)
```

```
> deseq <- DESeqDataSet(se, design= ~ 1)
> library(edgeR)
> edger <- DGEList(assays(se)$counts, group=rownames(colData(se)))
```

2.12 How to summarize junctions from a BAM file containing RNA-Seq reads

As sample data we use the *pasillaBamSubset* data package described in the introduction.

```
> library(pasillaBamSubset)
> un1 <- untreated1_chr4()  # single-end reads
> library(GenomicAlignments)
> reads1 <- readGAlignments(un1)
> reads1

GAlignments object with 204355 alignments and 0 metadata columns:
           seqnames strand       cigar    qwidth     start       end
              <Rle>  <Rle> <character> <integer> <integer> <integer>
       [1]     chr4      -         75M        75       892       966
       [2]     chr4      -         75M        75       919       993
       [3]     chr4      +         75M        75       924       998
       ...      ...    ...         ...       ...       ...       ...
  [204353]     chr4      +         75M        75   1348268   1348342
  [204354]     chr4      -         75M        75   1348449   1348523
  [204355]     chr4      -         75M        75   1350124   1350198
               width     njunc
           <integer> <integer>
       [1]        75         0
       [2]        75         0
       [3]        75         0
       ...       ...       ...
  [204353]        75         0
  [204354]        75         0
  [204355]        75         0
  -------
  seqinfo: 8 sequences from an unspecified genome
```

For each alignment, the aligner generated a CIGAR string that describes its "geometry", that is, the locations of insertions, deletions and junctions in the alignment. See the SAM Spec available on the SAMtools website for the details (http://samtools.sourceforge.net/).

The `summarizeJunctions()` function from the *GenomicAlignments* package can be used to summarize the junctions in `reads1`.

```
> junc_summary <- summarizeJunctions(reads1)
> junc_summary

GRanges object with 910 ranges and 3 metadata columns:
        seqnames          ranges strand |     score plus_score
           <Rle>       <IRanges>  <Rle> | <integer>  <integer>
    [1]     chr4      5246-11972      * |         3          1
```

```
    [2]     chr4     10346-10637      * |         1          1
    [3]     chr4     27102-27166      * |        13         11
    ...      ...             ...    ... .       ...        ...
  [908]     chr4 1333752-1346734      * |         1          0
  [909]     chr4 1334150-1347141      * |         1          1
  [910]     chr4 1334557-1347539      * |         1          0
        minus_score
          <integer>
    [1]           2
    [2]           0
    [3]           2
    ...         ...
  [908]           1
  [909]           0
  [910]           1
  -------
  seqinfo: 8 sequences from an unspecified genome
```

See `?summarizeJunctions` in the *GenomicAlignments* package for more information.

2.13 How to get the exon and intron sequences of a given gene

The exon and intron sequences of a gene are essentially the DNA sequences of the introns and exons of all known transcripts of the gene. The first task is to identify all transcripts associated with the gene of interest. Our sample gene is the human TRAK2 which is involved in regulation of endosome-to-lysosome trafficking of membrane cargo. The Entrez gene id is ‘66008’.

```
> trak2 <- "66008"
```

The *TxDb.Hsapiens.UCSC.hg19.knownGene* data package contains the gene model corresponding to the UCSC ‘Known Genes’ track.

```
> library(TxDb.Hsapiens.UCSC.hg19.knownGene)
> txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
```

The transcript ranges for all the genes in the gene model can be extracted with the `transcriptsBy` function from the *GenomicFeatures* package. They will be returned in a named *GRangesList* object containing all the transcripts grouped by gene. In order to keep only the transcripts of the TRAK2 gene we will subset the *GRangesList* object using the `[[` operator.

```
> library(GenomicFeatures)
> trak2_txs <- transcriptsBy(txdb, by="gene")[[trak2]]
> trak2_txs

GRanges object with 2 ranges and 2 metadata columns:
      seqnames              ranges strand |     tx_id     tx_name
         <Rle>           <IRanges>  <Rle> | <integer> <character>
  [1]     chr2 202241930-202316319      - |     12552  uc002uyb.4
  [2]     chr2 202259851-202316319      - |     12553  uc002uyc.2
```

```
  -------
  seqinfo: 93 sequences (1 circular) from hg19 genome
```

`trak2_txs` is a *GRanges* object with one range per transcript in the TRAK2 gene. The transcript names are stored in the `tx_name` metadata column. We will need them to subset the extracted intron and exon regions:

```
> trak2_tx_names <- mcols(trak2_txs)$tx_name
> trak2_tx_names

[1] "uc002uyb.4" "uc002uyc.2"
```

The exon and intron genomic ranges for all the transcripts in the gene model can be extracted with the `exonsBy` and `intronsByTranscript` functions, respectively. Both functions return a *GRangesList* object. Then we keep only the exon and intron for the transcripts of the TRAK2 gene by subsetting each *GRangesList* object by the TRAK2 transcript names.

Extract the exon regions:

```
> trak2_exbytx <- exonsBy(txdb, "tx", use.names=TRUE)[trak2_tx_names]
> elementNROWS(trak2_exbytx)

uc002uyb.4 uc002uyc.2
        16          8
```

... and the intron regions:

```
> trak2_inbytx <- intronsByTranscript(txdb, use.names=TRUE)[trak2_tx_names]
> elementNROWS(trak2_inbytx)

uc002uyb.4 uc002uyc.2
        15          7
```

Next we want the DNA sequences for these exons and introns. The `getSeq` function from the *Biostrings* package can be used to query a *BSgenome* object with a set of genomic ranges and retrieve the corresponding DNA sequences.

```
> library(BSgenome.Hsapiens.UCSC.hg19)
```

Extract the exon sequences:

```
> trak2_ex_seqs <- getSeq(Hsapiens, trak2_exbytx)
> trak2_ex_seqs

DNAStringSetList of length 2
[["uc002uyb.4"]] GCTGGGAGAGTGGCTCTCCTTTGGCTTCCCCAATTGTGTGGGGGCTGCCATT...
[["uc002uyc.2"]] GCTGGGAGAGTGGCTCTCCTTTGGCTTCCCCAATTGTGTGGGGGCTGCCATT...

> trak2_ex_seqs[["uc002uyb.4"]]

DNAStringSet object of length 16:
     width seq
 [1]   247 GCTGGGAGAGTGGCTCTCCTTTGGCTTCC...CGGACGACAGAGGATGCCGAACCACTCCA
 [2]   290 GTCATGACTGTCCAAAGTATGATAATCAC...CAATCACAGAGACTCGGAGAGCATCACTG
 [3]   195 ATGTCTGCTCCAATGAGGATCTCCCTGAA...CCTTGCTGAAGAGACTTTCCGTTACATGA
 ...   ... ...
[14]   267 GATCACAAACTCTGTATCACTGGCAGCAG...CATTACTTCAGCAGGTGGACCAGTTACAG
```

```
[15]   106 TTGCAACCGCCAACCCAGGAAAGTGCCTG...CCCTCTGACATCACTCAGGTTACCCCCAG
[16]  4012 CTCTGGGTTCCCTTCATTATCCTGTGGAA...TTAATAAACATGAGTAGCTTGAATTTTCA

> trak2_ex_seqs[["uc002uyc.2"]]

DNAStringSet object of length 8:
    width seq
[1]   247 GCTGGGAGAGTGGCTCTCCTTTGGCTTCCC...CGGACGACAGAGGATGCCGAACCACTCCA
[2]   290 GTCATGACTGTCCAAAGTATGATAATCACA...CAATCACAGAGACTCGGAGAGCATCACTG
[3]   195 ATGTCTGCTCCAATGAGGATCTCCCTGAAG...CCTTGCTGAAGAGACTTTCCGTTACATGA
[4]    77 TTCTAGGCACAGACAGGGTGGAGCAGATGA...TCGACATGGTTACACATCTCCTGGCAGAG
[5]   117 AGGGATCGTGATCTGGAACTCGCTGCTCGA...AGGAGCAATTGGGACAAGCCTTTGATCAA
[6]   210 GTTAATCAGCTGCAGCATGAGCTATGCAAG...AAGAAGAGAATATGGCTCTTCGATCCAAG
[7]    79 GCTTGTCACATAAAGACAGAAACTGTTACC...GCTTGTCAGCGACTGTGTTAAAGAACTTC
[8]   317 GTGAAACAAATGCTCAGATGTCCAGAATGA...AGATATCATGAATAAATACTTTCAAGTCA
```

... and the intron sequences:

```
> trak2_in_seqs <- getSeq(Hsapiens, trak2_inbytx)
> trak2_in_seqs

DNAStringSetList of length 2
[["uc002uyb.4"]] GTAAGAGTGCCTGGGAAATCTGGGGCCTCACTTCTTTCCTCAGCTATATTTT...
[["uc002uyc.2"]] GTGAGTATTAACATATTCTCTTTTGTACCTTTTTGGACAATTCTTTGGTAGG...

> trak2_in_seqs[["uc002uyb.4"]]

DNAStringSet object of length 15:
     width seq
 [1]  2892 GTAAGAGTGCCTGGGAAATCTGGGGCCTC...GTCTCCCACTTTTTTTTTTTTTTTTTAAG
 [2]  2001 GTGAGAAGAGTGTCTGGTTGAATATGGTA...TGTATTTGCTCCCTAAAAATCTATTTCAG
 [3]  1218 GTAATAAATCAGTAAGGGCCCTTACTAAG...TTTCCCCTTCCTTTGTTTTGCATATTCAG
 ...   ... ...
[13]  6308 GTGAGTATTTTTTTTACTCTTTTAGTTTG...CTATAAATAGTTGTTTTTAACTATATTAG
[14] 12819 GTAAGTCCAGTTTAATAAATATTGAAGTG...GATTCATTTACATAGACTCTCCTCTTTAG
[15] 30643 GTGAGTAAGCTGTCCGCGCAGAACCCGAA...GTTCTAGTCACTTGATGTTTTTGTTTTAG

> trak2_in_seqs[["uc002uyc.2"]]

DNAStringSet object of length 7:
    width seq
[1]  2057 GTGAGTATTAACATATTCTCTTTTGTACCT...AATTTAAAAAAATTTTTTTTGCTTCCAAG
[2]   564 GTACGTTCAACCTAATTGCCATTTTCCTTT...ATTGTCACATACTGATTTTTTTCTTGAAG
[3]  1022 GTAAGCCTTTGATCAAATGTCTGCAGTATG...CATGAAAATCAAGCATTTTATATGGACAG
[4]  1524 GTAGGAATATCTTTTCTTTCTCCAGTACAA...AAGAAAAGGTGTATTTGGTATTTTAACAG
[5]  6308 GTGAGTATTTTTTTTACTCTTTTAGTTTGT...CTATAAATAGTTGTTTTTAACTATATTAG
[6] 12819 GTAAGTCCAGTTTAATAAATATTGAAGTGC...GATTCATTTACATAGACTCTCCTCTTTAG
[7] 30643 GTGAGTAAGCTGTCCGCGCAGAACCCGAAC...GTTCTAGTCACTTGATGTTTTTGTTTTAG
```

2.14 How to get the CDS and UTR sequences of genes associated with colorectal cancer

In this *HOWTO* we extract the CDS and UTR sequences of genes involved in colorectal cancer. The workflow extends the ideas presented in the previous *HOWTO* and suggests an approach for identifying disease-related genes.

2.14.1 Build a gene list

We start with a list of gene or transcript ids. If you do not have pre-defined list one can be created with the *KEGG.db* and *KEGGgraph* packages. Updates to the data in the *KEGG.db* package are no longer available, however, the resource is still useful for identifying pathway names and ids.

Create a table of KEGG pathways and ids and search on the term 'cancer'.

```
> library(KEGG.db)
> pathways <- toTable(KEGGPATHNAME2ID)
> pathways[grepl("cancer", pathways$path_name, fixed=TRUE),]

    path_id                              path_name
370   05200                     Pathways in cancer
371   05202 Transcriptional misregulation in cancer
374   05205                Proteoglycans in cancer
375   05206                    MicroRNAs in cancer
376   05210                      Colorectal cancer
378   05212                      Pancreatic cancer
379   05213                     Endometrial cancer
381   05215                        Prostate cancer
382   05216                         Thyroid cancer
385   05219                         Bladder cancer
388   05222                 Small cell lung cancer
389   05223             Non-small cell lung cancer
390   05230     Central carbon metabolism in cancer
391   05231            Choline metabolism in cancer
```

Use the "05210" id to query the KEGG web resource (accesses the currently maintained data).

```
> library(KEGGgraph)
> dest <- tempfile()
> retrieveKGML("05200", "hsa", dest, "internal")
```

The suffix of the KEGG id is the Entrez gene id. The `translateKEGGID2GeneID` simply removes the prefix leaving just the Entrez gene ids.

```
> crids <- as.character(parseKGML2DataFrame(dest)[,1])
> crgenes <- unique(translateKEGGID2GeneID(crids))
> head(crgenes)

[1] "1630"  "836"   "842"   "1499"  "51384" "54361"
```

2.14.2 Identify genomic coordinates

The list of gene ids is used to extract genomic positions of the regions of interest. The Known Gene table from UCSC will be the annotation and is available as a *Bioconductor* package.

```
> library(TxDb.Hsapiens.UCSC.hg19.knownGene)
> txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
```

If an annotation is not available as a *Bioconductor* annotation package it may be available in *AnnotationHub*. Additionally, there are functions in *GenomicFeatures* which can retrieve data from UCSC and Ensembl to create a `TxDb`. See `?makeTxDbFromUCSC` for more information.

As in the previous *HOWTO* we need to identify the transcripts corresponding to each gene. The transcript id (or name) is used to isolate the UTR and coding regions of interest. This grouping of transcript by gene is also used to re-group the final sequence results.

The `transcriptsBy` function outputs both the gene and transcript identifiers which we use to create a map between the two. The `map` is a `CharacterList` with gene ids as names and transcript ids as the list elements.

```
> txbygene <- transcriptsBy(txdb, "gene")[crgenes] ## subset on colorectal genes
> map <- relist(unlist(txbygene, use.names=FALSE)$tx_id, txbygene)
> map

IntegerList of length 342
[["1630"]] 64962 64963 64964
[["836"]] 20202 20203 20204
[["842"]] 4447 4448 4449 4450 4451 4452
[["1499"]] 13582 13583 13584 13585 13586 13587 13589
[["51384"]] 29319 29320 29321
[["54361"]] 4634 4635
[["7471"]] 46151
[["7472"]] 31279 31280
[["7473"]] 63770
[["7474"]] 16089 16090 16091 16092
...
<332 more elements>
```

Extract the UTR and coding regions.

```
> cds <- cdsBy(txdb, "tx")
> threeUTR <- threeUTRsByTranscript(txdb)
> fiveUTR <- fiveUTRsByTranscript(txdb)
```

Coding and UTR regions may not be present for all transcripts specified in `map`. Consequently, the subset results will not be the same length. This length discrepancy must be taken into account when re-listing the final results by gene.

```
> txid <- unlist(map, use.names=FALSE)
> cds <- cds[names(cds) %in% txid]
> threeUTR <- threeUTR[names(threeUTR) %in% txid]
> fiveUTR <- fiveUTR[names(fiveUTR) %in% txid]
```

Note the different lengths of the subset regions.

```
> length(txid) ## all possible transcripts
[1] 1490
> length(cds)
[1] 1353
> length(threeUTR)
[1] 1308
> length(fiveUTR)
[1] 1339
```

These objects are `GRangesList`s with the transcript id as the outer list element.

```
> cds
GRangesList object of length 1353:
$`120`
GRanges object with 6 ranges and 3 metadata columns:
      seqnames          ranges strand |    cds_id    cds_name exon_rank
         <Rle>       <IRanges>  <Rle> | <integer> <character> <integer>
  [1]     chr1 1846720-1846746      + |       279        <NA>         1
  [2]     chr1 1847124-1847174      + |       280        <NA>         2
  [3]     chr1 1847880-1848054      + |       281        <NA>         3
  [4]     chr1 1848191-1848335      + |       282        <NA>         4
  [5]     chr1 1848413-1848513      + |       283        <NA>         5
  [6]     chr1 1848586-1848632      + |       284        <NA>         6
  -------
  seqinfo: 93 sequences (1 circular) from hg19 genome

$`1512`
GRanges object with 7 ranges and 3 metadata columns:
      seqnames            ranges strand |    cds_id    cds_name
         <Rle>         <IRanges>  <Rle> | <integer> <character>
  [1]     chr1 67666538-67666580      + |      4618        <NA>
  [2]     chr1 67672593-67672738      + |      4619        <NA>
  [3]     chr1 67685257-67685413      + |      4622        <NA>
  [4]     chr1 67702396-67702485      + |      4623        <NA>
  [5]     chr1 67705862-67705964      + |      4625        <NA>
  [6]     chr1 67721520-67721610      + |      4627        <NA>
  [7]     chr1 67724161-67724811      + |      4630        <NA>
      exon_rank
      <integer>
  [1]         5
  [2]         6
  [3]         7
  [4]         8
  [5]         9
  [6]        10
  [7]        11
  -------
```

```
  seqinfo: 93 sequences (1 circular) from hg19 genome

...
<1351 more elements>
```

2.14.3 Extract sequences from BSgenome

The `BSgenome` packages contain complete genome sequences for a given organism.

Load the `BSgenome` package for homo sapiens.

```
> library(BSgenome.Hsapiens.UCSC.hg19)
> genome <- BSgenome.Hsapiens.UCSC.hg19
```

Use `extractTranscriptSeqs` to extract the UTR and coding regions from the `BSgenome`. This function retrieves the sequences for an any `GRanges` or `GRangesList` (i.e., not just transcripts like the name implies).

```
> threeUTR_seqs <- extractTranscriptSeqs(genome, threeUTR)
> fiveUTR_seqs <- extractTranscriptSeqs(genome, fiveUTR)
> cds_seqs <- extractTranscriptSeqs(genome, cds)
```

The return values are `DNAStringSet` objects.

```
> cds_seqs

DNAStringSet object of length 1353:
       width seq                                     names
   [1]   546 ATGGGTCTTCAACAAGAA...TCAAGCTGATCCAGTAG 120
   [2]  1281 ATGGAAGAGTCAAAACAA...CACTCTTGGAAAAGTAG 1512
   [3]  1890 ATGAATCAGGTCACTATT...CACTCTTGGAAAAGTAG 1513
   ...   ... ...
[1351]  6012 ATGCAGCCCCCTTCACTG...GAGAGGGTAAAAAATAG 82649
[1352]   603 ATGTCAGCAGTTTGCTGT...TAACCCGCAAAGCCTGA 82652
[1353]  1944 ATGCAGCCCCCTTCACTG...CATTGGTGGGTGTTTGA 82653
```

Our final step is to collect the coding and UTR regions (currently organzied by transcript) into groups by gene id. The `relist` function groups the sequences of a `DNAStringSet` object into a `DNAStringSetList` object, based on the specified `skeleton` argument. The `skeleton` must be a list-like object and only its shape (i.e. its element lengths) matters (its exact content is ignored). A simple form of `skeleton` is to use a partitioning object that we make by specifying the size of each partition. The partitioning objects are different for each type of region because not all transcripts had a coding or 3' or 5' UTR region defined.

```
> lst3 <- relist(threeUTR_seqs, PartitioningByWidth(sum(map %in% names(threeUTR))))
> lst5 <- relist(fiveUTR_seqs, PartitioningByWidth(sum(map %in% names(fiveUTR))))
> lstc <- relist(cds_seqs, PartitioningByWidth(sum(map %in% names(cds))))
```

There are 239 genes in `map` each of which have 1 or more transcripts. The table of element lengths shows how many genes have each number of transcripts. For example, 47 genes have 1 transcript, 48 genes have 2 etc.

```
> length(map)

[1] 342

> table(elementNROWS(map))

 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 21 27 30 33 47
81 61 61 36 25 23 13  5  6  4  6  4  1  1  1  1  4  2  2  1  1  1  1  1
```

The lists of DNA sequences all have the same length as `map` but one or more of the element lengths may be zero. This would indicate that data were not available for that gene. The tables below show that there was at least 1 coding region available for all genes (i.e., none of the element lengths are 0). However, both the 3' and 5' UTR results have element lengths of 0 which indicates no UTR data were available for that gene.

```
> table(elementNROWS(lstc))

 1  2  3  4  5  6  7  8  9 10 11 12 14 15 16 17 18 20 26 30 35
83 68 67 36 19 23 10  6  7  4  2  2  3  1  2  1  4  1  1  1  1

> table(elementNROWS(lst3))

 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 20 26 30 35
 3 84 70 64 35 20 20 11  7  9  1  3  2  2  1  1  2  1  2  1  1  1  1

> names(lst3)[elementNROWS(lst3) == 0L] ## genes with no 3' UTR data

[1] "2255" "8823" "3443"

> table(elementNROWS(lst5))

 0  1  2  3  4  5  6  7  8  9 10 11 12 14 15 16 17 18 20 26 30 35
 4 82 66 67 35 19 23 10  6  7  4  3  2  3  1  1  1  4  1  1  1  1

> names(lst5)[elementNROWS(lst5) == 0L] ## genes with no 5' UTR data

[1] "2255"  "27006" "8823"  "3443"
```

2.15 How to create DNA consensus sequences for read group 'families'

The motivation for this *HOWTO* comes from a study which explored the dynamics of point mutations. The mutations of interest exist with a range of frequencies in the control group (e.g., 0.1% - 50%). PCR and sequencing error rates make it difficult to identify low frequency events (e.g., $<$ 20%).

When a library is prepared with Nextera, random fragments are generated followed by a few rounds of PCR. When the genome is large enough, reads aligning to the same start position are likely descendant from the same template fragment and should have identical sequences.

The goal is to elimininate noise by grouping the reads by common start position and discarding those that do not exceed a certain threshold within each family. A new consensus sequence will be created for each read group family.

2.15.1 Sort reads into groups by start position

Load the BAM file into a GAlignments object.

```
> library(Rsamtools)
> bamfile <- system.file("extdata", "ex1.bam", package="Rsamtools")
> param <- ScanBamParam(what=c("seq", "qual"))
> library(GenomicAlignments)
> gal <- readGAlignments(bamfile, use.names=TRUE, param=param)
```

Use the `sequenceLayer` function to *lay* the query sequences and quality strings on the reference.

```
> qseq <- setNames(mcols(gal)$seq, names(gal))
> qual <- setNames(mcols(gal)$qual, names(gal))
> qseq_on_ref <- sequenceLayer(qseq, cigar(gal),
+                              from="query", to="reference")
> qual_on_ref <- sequenceLayer(qual, cigar(gal),
+                              from="query", to="reference")
```

Split by chromosome.

```
> qseq_on_ref_by_chrom <- splitAsList(qseq_on_ref, seqnames(gal))
> qual_on_ref_by_chrom <- splitAsList(qual_on_ref, seqnames(gal))
> pos_by_chrom <- splitAsList(start(gal), seqnames(gal))
```

For each chromosome generate one GRanges object that contains unique alignment start positions and attach 3 metadata columns to it: the number of reads, the query sequences, and the quality strings.

```
> gr_by_chrom <- lapply(seqlevels(gal),
+   function(seqname)
+   {
+     qseq_on_ref2 <- qseq_on_ref_by_chrom[[seqname]]
+     qual_on_ref2 <- qual_on_ref_by_chrom[[seqname]]
+     pos2 <- pos_by_chrom[[seqname]]
+     qseq_on_ref_per_pos <- split(qseq_on_ref2, pos2)
+     qual_on_ref_per_pos <- split(qual_on_ref2, pos2)
+     nread <- elementNROWS(qseq_on_ref_per_pos)
+     gr_mcols <- DataFrame(nread=unname(nread),
+                           qseq_on_ref=unname(qseq_on_ref_per_pos),
+                           qual_on_ref=unname(qual_on_ref_per_pos))
+     gr <- GRanges(Rle(seqname, nrow(gr_mcols)),
+                   IRanges(as.integer(names(nread)), width=1))
+     mcols(gr) <- gr_mcols
+     seqlevels(gr) <- seqlevels(gal)
+     gr
+   })
```

Concatenate all the GRanges objects obtained in (4) together in 1 big GRanges object:

```
> gr <- do.call(c, gr_by_chrom)
> seqinfo(gr) <- seqinfo(gal)
```

‘gr’ is a GRanges object that contains unique alignment start positions:

```
> gr[1:6]

GRanges object with 6 ranges and 3 metadata columns:
      seqnames    ranges strand |     nread
         <Rle> <IRanges>  <Rle> | <integer>
  [1]     seq1         1      * |         1
  [2]     seq1         3      * |         1
  [3]     seq1         5      * |         1
  [4]     seq1         6      * |         1
  [5]     seq1         9      * |         1
  [6]     seq1        13      * |         2
                                        qseq_on_ref
                                 <DNAStringSetList>
  [1]                        CACTAGTGGCTCATTGTAAA..
  [2]                        CTAGTGGCTCATTGTAAATG..
  [3]                        AGTGGCTCATTGTAAATGTG..
  [4]                        GTGGCTCATTGTAATTTTTT..
  [5]                        GCTCATTGTAAATGTGTGGT..
  [6] ATTGTAAATGTGTGGTTTAA..,ATTGTAAATGTGTGGTTTAA..
                                        qual_on_ref
                                   <BStringSetList>
  [1]                        <<<<<<<<<<<<<<<;<<<<..
  [2]                        <<<<<<<<<<0<<<<655<<..
  [3]                        <<<<<<<<<<<7;71<<;<;..
  [4]                        (-&----,----)-)-),'-..
  [5]                        <<<<<<<<<<<<<<<;<;7<..
  [6] <<<<<<<<;<<<8<<<<<;8..,<<<<<;<<<;<;<<<<<<<<..
  -------
  seqinfo: 2 sequences from an unspecified genome
```

Look at qseq_on_ref and qual_on_ref.

```
> qseq_on_ref

DNAStringSet object of length 3271:
       width seq                                    names
   [1]    36 CACTAGTGGCTCATTGTAAATGTGTGGTTTAACTCG   B7_591:4:96:693:509
   [2]    35 CTAGTGGCTCATTGTAAATGTGTGGTTTAACTCGT    EAS54_65:7:152:36...
   [3]    35 AGTGGCTCATTGTAAATGTGTGGTTTAACTCGTCC    EAS51_64:8:5:734:57
   ...   ... ...
[3269]    35 TTTTTTCTTTTTTTTTTTTTTTTTTTTGCATGCCA    EAS139_11:7:50:12...
[3270]    35 TTTTTTTTTTTTTTTTTTTTTTTGCATGCCAGAAA    EAS54_65:3:320:20...
[3271]    35 TTTTTTTTTTTTTTTTTTTTTTTCATGCCAGAAAA    EAS114_26:7:37:79...

> qual_on_ref

PhredQuality object of length 3271:
       width seq                                    names
   [1]    36 <<<<<<<<<<<<<<<;<<<<<<<<<5<<<<<;:<;7   B7_591:4:96:693:509
   [2]    35 <<<<<<<<<<0<<<<655<<7<<<:9<<3/:<6):    EAS54_65:7:152:36...
   [3]    35 <<<<<<<<<<<7;71<<;<;;<7;<<3;);3*8/5    EAS51_64:8:5:734:57
   ...   ... ...
```

```
[3269]     35 <<<<,<&<7<<<<<<<<<<<<<<<<<<<<<<<<<<    EAS139_11:7:50:12...
[3270]     35 +'''/<<<<7:;+<;::<<<;;<<<<<<<<<<<<<    EAS54_65:3:320:20...
[3271]     35 3,,,===6===<===<;=====-============    EAS114_26:7:37:79...
```

2 reads align to start position 13. Let's have a close look at their sequences:

```
> mcols(gr)$qseq_on_ref[[6]]

DNAStringSet object of length 2:
    width seq                                      names
[1]    35 ATTGTAAATGTGTGGTTTAACTCGTCCCTGGCCCA      EAS56_61:6:18:467...
[2]    36 ATTGTAAATGTGTGGTTTAACTCGTCCATGGCCCAG     EAS114_28:5:296:3...
```

and their qualities:

```
> mcols(gr)$qual_on_ref[[6]]

PhredQuality object of length 2:
    width seq                                      names
[1]    35 <<<<<<<<;<<<8<<<<<;8:;6/686&;(16666      EAS56_61:6:18:467...
[2]    36 <<<<<;<<<;<;<<<<<<<<<<<8<8<3<8;<;<0;     EAS114_28:5:296:3...
```

Note that the sequence and quality strings are those projected to the reference so the first letter in those strings are on top of start position 13, the 2nd letter on top of position 14, etc...

2.15.2 Remove low frequency reads

For each start position, remove reads with and under-represented sequence (e.g. threshold = 20% for the data used here which is low coverage). A unique number is assigned to each unique sequence. This will make future calculations easier and a little bit faster.

```
> qseq_on_ref <- mcols(gr)$qseq_on_ref
> tmp <- unlist(qseq_on_ref, use.names=FALSE)
> qseq_on_ref_id <- relist(match(tmp, tmp), qseq_on_ref)
```

Quick look at 'qseq_on_ref_id': It's an IntegerList object with the same length and "shape" as 'qseq_on_ref'.

```
> qseq_on_ref_id

IntegerList of length 1934
[[1]] 1
[[2]] 2
[[3]] 3
[[4]] 4
[[5]] 5
[[6]] 6 7
[[7]] 8
[[8]] 9
[[9]] 10 11
[[10]] 12
```

```
...
<1924 more elements>
```

Remove the under represented ids from each list element of ‘qseq_on_ref_id’:

```
> qseq_on_ref_id2 <- endoapply(qseq_on_ref_id,
+     function(ids) ids[countMatches(ids, ids) >= 0.2 * length(ids)])
```

Remove corresponding sequences from ‘qseq_on_ref’:

```
> tmp <- unlist(qseq_on_ref_id2, use.names=FALSE)
> qseq_on_ref2 <- relist(unlist(qseq_on_ref, use.names=FALSE)[tmp],
+                        qseq_on_ref_id2)
```

2.15.3 Create a consensus sequence for each read group family

Compute 1 consensus matrix per chromosome:

```
> split_factor <- rep.int(seqnames(gr), elementNROWS(qseq_on_ref2))
> qseq_on_ref2 <- unlist(qseq_on_ref2, use.names=FALSE)
> qseq_on_ref2_by_chrom <- splitAsList(qseq_on_ref2, split_factor)
> qseq_pos_by_chrom <- splitAsList(start(gr), split_factor)
> cm_by_chrom <- lapply(names(qseq_pos_by_chrom),
+     function(seqname)
+         consensusMatrix(qseq_on_ref2_by_chrom[[seqname]],
+                         as.prob=TRUE,
+                         shift=qseq_pos_by_chrom[[seqname]]-1,
+                         width=seqlengths(gr)[[seqname]]))
> names(cm_by_chrom) <- names(qseq_pos_by_chrom)
```

’cm_by_chrom’ is a list of consensus matrices. Each matrix has 17 rows (1 per letter in the DNA alphabet) and 1 column per chromosome position.

```
> lapply(cm_by_chrom, dim)

$seq1
[1]   18 1575

$seq2
[1]   18 1584
```

Compute the consensus string from each consensus matrix. We’ll put "+" in the strings wherever there is no coverage for that position, and "N" where there is coverage but no consensus.

```
> cs_by_chrom <- lapply(cm_by_chrom,
+     function(cm) {
+         ## need to "fix" 'cm' because consensusString()
+         ## doesn't like consensus matrices with columns
+         ## that contain only zeroes (e.g., chromosome
+         ## positions with no coverage)
+         idx <- colSums(cm) == 0L
```

```
+         cm["+", idx] <- 1
+         DNAString(consensusString(cm, ambiguityMap="N"))
+     })
```

The new consensus strings.

```
> cs_by_chrom

$seq1
1575-letter DNAString object
seq: NANTAGNNNCTCANTTTAAANNTTTNTTTTTN...AATNATANNTTTNTTNTTTNTCTGNAC+++++

$seq2
1584-letter DNAString object
seq: ++++++++++++++++++++++++++++++++...NNNANANANANCTNNA++++++++++++++++
```

2.16 How to compute binned averages along a genome

In some applications (e.g. visualization), there is the need to compute the average of a variable defined along a genome (a.k.a. genomic variable) for a set of predefined fixed-width regions (sometimes called "bins"). The genomic variable is typically represented as a named `RleList` object with one list element per chromosome. One such example is coverage. Here we create an artificial genomic variable:

```
> library(BSgenome.Scerevisiae.UCSC.sacCer2)
> set.seed(55)
> my_var <- RleList(
+     lapply(seqlengths(Scerevisiae),
+         function(seqlen) {
+             tmp <- sample(50L, seqlen, replace=TRUE) %/% 50L
+             Rle(cumsum(tmp - rev(tmp)))
+         }
+     ),
+     compress=FALSE)
> my_var

RleList of length 18
$chrI
integer-Rle of length 230208 with 9197 runs
  Lengths:  6 17 12 12 13 38 15 24 24 25 ... 24 24 15 38 13 12 12 17  7
  Values :  0  1  0  1  2  3  4  3  4  3 ...  4  3  4  3  2  1  0  1  0

$chrII
integer-Rle of length 813178 with 31826 runs
  Lengths: 35 84 50 44  7 67 18  8  7 27 ...  8 18 67  7 44 50 84 35  1
  Values : -1 -2 -1  0  1  0  1  2  1  2 ...  2  1  0  1  0 -1 -2 -1  0

$chrIII
integer-Rle of length 316617 with 12601 runs
  Lengths: 64 16  1 63 48 20 32 43 12 68 ... 12 43 32 20 48 63  1 16 65
  Values :  0  1  0  1  0  1  0  1  2  1 ...  2  1  0  1  0  1  0  1  0
```

```
$chrIV
integer-Rle of length 1531919 with 60615 runs
  Lengths:   2  19  38  14  10   8  20 ...  20   8  10  14  38  19   3
  Values :   0  -1  -2  -1  -2  -3  -2 ...  -2  -3  -2  -1  -2  -1   0

$chrV
integer-Rle of length 576869 with 22235 runs
  Lengths:  10  69  31   7   3   1   5 ...   5   1   3   7  31  69  11
  Values :   0   1   2   1   2   1   0 ...   0   1   2   1   2   1   0

...
<13 more elements>
```

Use the `tileGenome` function to create a set of bins along the genome.

```
> bins <- tileGenome(seqinfo(Scerevisiae), tilewidth=100,
+                     cut.last.tile.in.chrom=TRUE)
```

Compute the binned average for `my_var`:

```
> binnedAverage(bins, my_var, "binned_var")

GRanges object with 121639 ranges and 1 metadata column:
           seqnames    ranges strand | binned_var
              <Rle> <IRanges>  <Rle> |  <numeric>
       [1]     chrI     1-100      * |       1.77
       [2]     chrI   101-200      * |       3.34
       [3]     chrI   201-300      * |       3.22
       ...      ...       ...    ... .        ...
  [121637]  2micron 6101-6200      * |  -1.000000
  [121638]  2micron 6201-6300      * |  -0.750000
  [121639]  2micron 6301-6318      * |  -0.555556
  -------
  seqinfo: 18 sequences (2 circular) from sacCer2 genome
```

The bin size can be modified with the `tilewidth` argument to `tileGenome`. See `?binnedAverage` for additional examples.

3 Session Information

```
R version 4.0.3 (2020-10-10)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 18.04.5 LTS

Matrix products: default
BLAS:   /home/biocbuild/bbs-3.12-bioc/R/lib/libRblas.so
LAPACK: /home/biocbuild/bbs-3.12-bioc/R/lib/libRlapack.so

locale:
```

```
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] parallel  stats4    stats     graphics  grDevices utils
[7] datasets  methods   base

other attached packages:
 [1] BSgenome.Scerevisiae.UCSC.sacCer2_1.4.0
 [2] KEGGgraph_1.50.0
 [3] KEGG.db_3.2.4
 [4] BSgenome.Hsapiens.UCSC.hg19_1.4.3
 [5] BSgenome_1.58.0
 [6] rtracklayer_1.50.0
 [7] edgeR_3.32.0
 [8] limma_3.46.0
 [9] DESeq2_1.30.0
[10] AnnotationHub_2.22.0
[11] BiocFileCache_1.14.0
[12] dbplyr_1.4.4
[13] TxDb.Athaliana.BioMart.plantsmart22_3.0.1
[14] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
[15] TxDb.Dmelanogaster.UCSC.dm3.ensGene_3.2.2
[16] GenomicFeatures_1.42.0
[17] AnnotationDbi_1.52.0
[18] GenomicAlignments_1.26.0
[19] Rsamtools_2.6.0
[20] Biostrings_2.58.0
[21] XVector_0.30.0
[22] SummarizedExperiment_1.20.0
[23] Biobase_2.50.0
[24] MatrixGenerics_1.2.0
[25] matrixStats_0.57.0
[26] pasillaBamSubset_0.27.0
[27] GenomicRanges_1.42.0
[28] GenomeInfoDb_1.26.0
[29] IRanges_2.24.0
[30] S4Vectors_0.28.0
[31] BiocGenerics_0.36.0
[32] BiocStyle_2.18.0

loaded via a namespace (and not attached):
 [1] bitops_1.0-6                  bit64_4.0.5
 [3] RColorBrewer_1.1-2            progress_1.2.2
 [5] httr_1.4.2                    tools_4.0.3
 [7] R6_2.4.1                      colorspace_1.4-1
 [9] DBI_1.1.0                     tidyselect_1.1.0
```

```
 [11] prettyunits_1.1.1             bit_4.0.4
 [13] curl_4.3                      compiler_4.0.3
 [15] graph_1.68.0                  xml2_1.3.2
 [17] DelayedArray_0.16.0           bookdown_0.21
 [19] scales_1.1.1                  genefilter_1.72.0
 [21] askpass_1.1                   rappdirs_0.3.1
 [23] stringr_1.4.0                 digest_0.6.27
 [25] rmarkdown_2.5                 pkgconfig_2.0.3
 [27] htmltools_0.5.0               fastmap_1.0.1
 [29] rlang_0.4.8                   RSQLite_2.2.1
 [31] shiny_1.5.0                   generics_0.0.2
 [33] BiocParallel_1.24.0           dplyr_1.0.2
 [35] VariantAnnotation_1.36.0      RCurl_1.98-1.2
 [37] magrittr_1.5                  GenomeInfoDbData_1.2.4
 [39] Matrix_1.2-18                 Rcpp_1.0.5
 [41] munsell_0.5.0                 lifecycle_0.2.0
 [43] stringi_1.5.3                 yaml_2.2.1
 [45] zlibbioc_1.36.0               grid_4.0.3
 [47] blob_1.2.1                    promises_1.1.1
 [49] crayon_1.3.4                  lattice_0.20-41
 [51] splines_4.0.3                 annotate_1.68.0
 [53] hms_0.5.3                     locfit_1.5-9.4
 [55] knitr_1.30                    pillar_1.4.6
 [57] geneplotter_1.68.0            biomaRt_2.46.0
 [59] XML_3.99-0.5                  glue_1.4.2
 [61] BiocVersion_3.12.0            evaluate_0.14
 [63] BiocManager_1.30.10           vctrs_0.3.4
 [65] httpuv_1.5.4                  gtable_0.3.0
 [67] openssl_1.4.3                 purrr_0.3.4
 [69] assertthat_0.2.1              ggplot2_3.3.2
 [71] xfun_0.18                     mime_0.9
 [73] xtable_1.8-4                  later_1.1.0.1
 [75] survival_3.2-7                tibble_3.0.4
 [77] memoise_1.1.0                 ellipsis_0.3.1
 [79] interactiveDisplayBase_1.28.0
```

References

Michael Lawrence, Wolfgang Huber, Hervé Pagès, Patrick Aboyoun, Marc Carlson, Robert Gentleman, Martin T. Morgan, and Vincent J. Carey. Software for computing and annotating genomic ranges. *PLOS Computational Biology*, 4(3), 2013.