BioC for HTS - PDCB topic
Infrastructure and Input/Output 01

LCG Leonardo Collado Torres
lcollado@wintergenomics.com – lcollado@ibt.unam.mx

September 29th, 2010
Overview

ShortRead

SAM format

Rsamtools

Exercises
Infra-I/O

▶ This is the beginning of the *infrastructure and input/output* section of the course.

▶ Our goals: to learn how to read into R HTS data files, manipulate them and manipulate the information they contain.

▶ Today is mostly about reading in files and filtering reads we don’t want.
Today’s packages

- You should have them installed already, but if you don’t then please do so:

  > source("http://bioconductor.org/biocLite.R")
  > biocLite(c("ShortRead", "Rsamtools"))
HTS data formats

► Which HTS data formats are you familiar with or have heard about?

► A

NAGAGGCCAGGCCATCTACCACCTTTTGTGGAAATTTTGCTCTTTCAAC
+HWUSI-EAS636_0001:1:1:0:114#0/1
DOVYUQUYWSWTWYYYYYYYYYYYYYYYYYTPLSYWWRJRYYYYWVTV
CGGAAGAGCGGTTCCAAGGAATGCCGAGATCGGAAGAGCGGTTCAGCAT
+HWUSI-EAS636_0001:1:1:1:552#0/1
aaaaaaaK_Y^_b_ZaaPIXa_VZWRNHZ^LHUHRRPUPVJIRQWQYXB
AGCGCATCTTGCCTATGTGCAGCAGCAGCGGTAGCCTTAACCTGATGC

► B
HTS data formats

```
HWUSI-EAS636  1    4    45    849   1900  0    1
GACTTGGTCACTAAATACTTCAAACCAAATAGGGCA
abbabbbababbababababaaaaaaaaaabaa]`aa ECK12.fasta  113

F    36   146
HWUSI-EAS636  1    4    101   43     603  0    1
CTTAGGTCACTAAATACTTCAAACCAAATAGGGCATA
Z_b]J_`Zaa]]`bbabba`_`aaabaa_`TT`_W ECK12.fasta  115

F    36   146
HWUSI-EAS636  1    4    109  1181   363  0    1
CTTAGGTCACTAAATACTTCAAACCAAATAGGCCATA
aa`baa`aaabaaabaaaaaaa``aaaaa__``_aaa ECK12.fasta  115

F    36   146
```
HTS data formats

```gff
#gff-version 2
#date 2010-09-13
Ecoli rtracklayer sequence 16 16 3 - 
1.4
Ecoli rtracklayer sequence 38 38 24 - 
2.4
Ecoli rtracklayer sequence 50 50 6 - 
3.4

▶ D
HWUSI-EAS636:8:120:1791:562#0/1 - gi|49175990|ref|NC_000913.2|
1753519 GTCGGACTGTAGAACTCT ::::868;>>>:>>B 0
0:A>T,15:T>C,17:T>G
HWUSI-EAS636:8:120:1791:393#0/1 - gi|49175990|ref|NC_000913.2|
2399840 TCGGACTGTAGAACTCTG 9>7@1B;<8@AA8A8AAB 0
2:T>C,3:G>T,15:T>G
HWUSI-EAS636:8:120:1791:1802#0/1 + gi|49175990|ref|
NC_000913.2| 1132065 GTTCAGAGTTCTACAGTC B;;>9>4;;;;=67;?
0 4:G>A,6:T>A,16:C>T
```

7 / 70
# HTS data formats

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Forward</th>
<th>Reverse</th>
<th>Variants</th>
</tr>
</thead>
<tbody>
<tr>
<td>HWUSI-EAS636:8:120:1791:1350#0/1 -</td>
<td>gi</td>
<td>49175990</td>
<td>ref</td>
</tr>
<tr>
<td>NC_000913.2</td>
<td>1753520 TCGGACTGTAAGACTCTG &lt;:A8866?;8@:6&gt;&lt;?B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0:A&gt;G,1:A&gt;T,16:T&gt;C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HWUSI-EAS636_0009:8:120:16043:16103#0/1 +</td>
<td>gi</td>
<td>49175990</td>
<td>ref</td>
</tr>
<tr>
<td>NC_000913.2</td>
<td>4091324 GCCGAATTAGATGGC B############### 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1:T&gt;C,5:G&gt;A,6:A&gt;T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HWUSI-EAS636_0009:8:120:16043:16103#0/2 -</td>
<td>gi</td>
<td>49175990</td>
<td>ref</td>
</tr>
<tr>
<td>NC_000913.2</td>
<td>4091687 TTTTTGCTTCTTTTTT #iddles#@@23&lt; 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0:G&gt;T,3:C&gt;T,7:A&gt;T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HWUSI-EAS636_0009:8:120:16061:14457#0/1 +</td>
<td>gi</td>
<td>49175990</td>
<td>ref</td>
</tr>
<tr>
<td>NC_000913.2</td>
<td>584209 GCCACCGAGTTAAAA C############### 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11:C&gt;A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HWUSI-EAS636_0009:8:120:16061:14457#0/2 -</td>
<td>gi</td>
<td>49175990</td>
<td>ref</td>
</tr>
<tr>
<td>NC_000913.2</td>
<td>584573 CTGAGAGTTGTACAT #iddles### 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0:G&gt;T,3:C&gt;A,13:C&gt;T</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
HTS data formats

```
track name="R Track" type=bedGraph
chr_gi|49175990|ref|NC_000913.2| 18 45 2
chr_gi|49175990|ref|NC_000913.2| 81 95 0
chr_gi|49175990|ref|NC_000913.2| 95 104 2
```

- **G**

  B7_591:4:96:693:509 73 seq1 1 99 36M * 0 0
  CACTAGTGGCTCATTGTAAATGTGTGGTTTAACTCG <<<<<<<<<<<<<<<<;<<<<<<<<<5<<<<<<;:<;7
  MF:i:18 Aq:i:73 NM:i:0 UQ:i:0 H0:i:1 H1:i:0
  EAS54_65:7:152:368:113 73 seq1 3 99 35M * 0 0
  CTAGTGGCTCATTGTAAATGTGTGGTTAACTCGT <<<<<<<<<<<<<<<<<<<<<5<<<<<<;:<<;6:
  MF:i:18 Aq:i:66 NM:i:0 UQ:i:0 H0:i:1 H1:i:0
  EAS51_64:8:5:734:57 137 seq1 5 99 35M * 0 0
  AGTGGCTCATTGTAAATGTGTGGTTAACTCGTCC <<<<<<<<<<<<<<<<7;71<<;<<;7<<;<<3;>:3*8/5
  MF:i:18 Aq:i:66 NM:i:0 UQ:i:0 H0:i:1 H1:i:0

- **H**
HTS data formats

HWI-EAS88_4_1_6_505_934 ChrA 1 + 0 0 15 15 15 1 12 0 1 35  
aaagttagagaagtttgacttttgtaggcaccatc --------------------------)))))))###
HWI-EAS88_4_1_7_163_963 ChrA 1 + 0 0 22 22 22 0 0 1 0 35  
aaagttagagaagtttgacttttgtaggcaccatc --------------------------)))))))###

I

HWI-EAS88 3 2 1 451 945 CCAGAGCCCCCGCTCACTCCTGAACCAGTCTCTC  
YQMIMIMMLMMIGIGMFICMFFFFIMMHiIHAAGAH NM N  
HWI-EAS88 3 2 1 409 991 AGCCTCCCTCTTTCTGAATATACGGCAGAGCTGT  
ZXZUYXZQYYUXZXYZZYZZXZIMFHXQSUPPO NM Y  
HWI-EAS88 3 2 1 451 939 ACCAAAACACCACATACACGAGCAACACACGTAC  
LGDHLILLLLLLLLLIGFLALDIFDILLHFIIEAECAE NM N

J

I'm a HTS data file from your imagination :)
Answers

- A fastq
- B sorted
- C gff version 2
- D bowtie single end
- E bowtie paired end
- F bed
- G SAM
- H maq
- I export
- J :O
The ShortRead package

- It’s one of the first BioC packages related to HTS data
- Has been the basic input/output package for HTS data
- It can read solexa, fastq, bowtie, and maq files. It can also read in other types of alignments.
- With it we can explore the quality of our reads/alignments, create a report and filter out reads.
- Current model: read all the reads into RAM and then manipulate them.
Our first steps with ShortRead

- Lets get into ShortRead!
- SR was originally designed to read in files from the Solexa set of directories.
- Lets look at the example data. Where is it for you?
  ```r
  > library(ShortRead)
  > exptPath <- system.file("extdata",
    +   package = "ShortRead")
  > sp <- SolexaPath(exptPath)
  > sp
  ```
- For SR to recognize the path, we need to use `SolexaPath`:
  ```r
  > sp <- SolexaPath(exptPath)
  > sp
  ```
Our first steps with ShortRead

class: SolexaPath
experimentPath: /usr/local/lib64/R/library/ShortRead/extdata
dataPath: Data
scanPath: NA
imageAnalysisPath: C1-36Firecrest
baseCallPath: Bustard
analysisPath: GERALD

Next, we can use some functions to find the path for several important files:

> imageAnalysisPath(sp)

[1] "/usr/local/lib64/R/library/ShortRead/extdata/Data/C1-36Firecrest"

> analysisPath(sp)
Our first steps with ShortRead

[1] "/usr/local/lib64/R/library/ShortRead/extdata/Data/

▶ However, that isn’t that interesting for us. We want to read in data! For example, an export file.

```r
> aln <- readAligned(sp, "s_2_export.txt")
> aln

class: AlignedRead
length: 1000 reads; width: 35 cycles
chromosome: NM NM ... chr5.fa 29:255:255
position: NA NA ... 71805980 NA
strand: NA NA ... + NA
alignQuality: NumericQuality
alignData varLabels: run lane ... filtering contig
```

> class(aln)
Our first steps with ShortRead

[1] "AlignedRead"
attr(,"package")
[1] "ShortRead"

- AlignedRead objects are the main type of objects in SR. Multiple functions to access parts of it exist.
- For example, how would you extract the positions for all reads?
As the names imply, we can extract the positions with:

```r
> summary(position(aln))
```

```
   Min. 1st Qu.  Median    Mean
11940  34710000  73390000  74160000
```

```
  3rd Qu.  Max.  NA's
108500000 195500000   594
```

Why do we have NAs?

Some other useful accesors are:

```r
> table(strand(aln))
```

```
   +  -  *
203 203  0
```
AlignedRead

> unique(width(aln))

[1] 35

> alignQuality(aln)

class: NumericQuality

quality: 0 0 ... 55 0 (1000 total)

> summary(quality(alignQuality(aln)))

    Min. 1st Qu.  Median    Mean 3rd Qu.  Max.
 0.0000 0.0000  0.0000  17.040  37.000 72.000

> length(aln)

[1] 1000
**AlignedRead**

```r
> head(table(chromosome(aln)))

0:0:187 0:0:19 0:0:21 0:0:25 0:0:255
 1 1 1 2 1

0:0:85
 1

> head(id(aln))

A BStringSet instance of length 6

width seq
[1] 0
[2] 0
[3] 0
[4] 0
```
## AlignedRead

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[5]</td>
<td>0</td>
</tr>
<tr>
<td>[6]</td>
<td>0</td>
</tr>
</tbody>
</table>
Quick exercise

- Lets assume that the 5’ end of our reads corresponds to transcription start sites.
- Get the TSSs positions.
- What is the TSSs for read number 10 in our aln object?
- Remember:
  
  ```r
  > summary(position(aln))[7]
  NA's
  594
  ```
Solution

- Lets take advantage of how R works by using vectors.
  ```r
  > idx <- which(is.na(position(aln)) ==
  +   FALSE)
  > neg <- which(strand(aln)[idx] ==
  +   "-")
  > tss <- position(aln)[idx]
  > tss[neg] <- tss[neg] + width(aln)[idx][neg] -
  +   1
  ```

- For the second part:
  ```r
  > tenth <- head(position(aln), 10)
  > tenth
  ```
Solution

```
[1]   NA   NA   NA   NA   NA
[5]   NA   NA 69345321 54982866
[9]   NA 80537786

> tenth <- length(which(is.na(tenth) == FALSE))
> tenth

[1] 3

> tss[tenth]

[1] 80537820

▶ Is the answer correct?

> aln[10]
```

```
23 / 70
```
Solution

class: AlignedRead
length: 1 reads; width: 35 cycles
chromosome: chr12.fa
position: 80537786
strand: -
alignQuality: NumericQuality
alignData varLabels: run lane ... filtering contig

> tss[tenth] == 80537786 + 35 - 1

[1] TRUE
Reading fastq files

▶ Before we continue with alignment files, SR is also capable of reading fastq files.
▶ Let's read the example file:
  ```
  > args(readFastq)
  function (dirPath, pattern = character(0), ...) NULL
  > sread <- readFastq(analysisPath(sp),
  + pattern = "sequence.txt")
  > class(sread)
  [1] "ShortReadQ"
  attr(,"package")
  [1] "ShortRead"
  >
  ▶ What did analysisPath do for us?
In addition to AlignedRead, ShortReadQ objects completes the family of main objects in SR.

```r
> sread
```

- class: ShortReadQ
- length: 256 reads; width: 36 cycles

Similar to AlignedRead objects, we can access parts of the information:

```r
> head(id(sread))
```
ShortReadQ

A BStringSet instance of length 6

width seq
[1]  24 HWI-EAS88_1_1_1_1001_499
[2]  23 HWI-EAS88_1_1_1_898_392
[3]  23 HWI-EAS88_1_1_1_922_465
[4]  23 HWI-EAS88_1_1_1_895_493
[5]  23 HWI-EAS88_1_1_1_953_493
[6]  23 HWI-EAS88_1_1_1_868_763

> head(quality(sread))
class: SFastqQuality

quality:

A BStringSet instance of length 6

width seq

[1] 36 ]]]]]]]]]]]]]]Y][...]]]]]VCHVMPLAS
[5] 36 ]]]]]]]]]]]]]]]...]]]]]MJUJVLSS

> length(sread)

[1] 256

> table(width(sread))
ShortReadQ

36
256

> head(sread(sread))

A DNAStringSet instance of length 6

width seq

[1] 36 GGACTTTTGTAGGAT...TTCCTTCTCCTGT
[2] 36 GATTTCTTACCTAT...AACAGCATCGGAC
[3] 36 GCGGTGGTCTATAG...TATCAATTTGGGT
[4] 36 GTTACCATGATGTT...TTTGGAGGTAAAA
[5] 36 GTATGTTTCTCCTG...TTCTTGAAGGCTT
[6] 36 GTTCTCTAAAAACC...CCCCTTCGGGGC

> narrow(sread, start = 1, end = 10)
ShortReadQ

class: ShortReadQ
length: 256 reads; width: 10 cycles
Exercise

- Which are different dinucleotides in our reads? Only base 1 and 2 of our reads.
- What are the frequencies of the different dinucleotides?
- Coercion functions such as `as.character` can be useful :) You might need to check the help of:

```r
> `?` (BStringSet)
```
Solution 1

- Lets use the sread, narrow, as.character and table functions:
  
  ```r
  > first2 <- sread(narrow(sread, start = 1, width = 2))
  > head(first2)
  
  A DNAStringSet instance of length 6
  
  width  seq
  [1]  2  GG
  [2]  2  GA
  [3]  2  GC
  [4]  2  GT
  [5]  2  GT
  [6]  2  GT
  ```
Solution 1

```r
> first2 <- as.character(first2)
> table(first2)

   GA GC GG GT
first2  61 42  51 102
```
Solution II

- While the above solution was fine, it did involve changing between types of objects.
- Let's use the `dinucleotideFrequency` function:
  ```r
  > dinuc <- dinucleotideFrequency(sread(narrow(sread, + start = 1, width = 2)))
  > dinuc[1, ]
  # Output:
  # AA  AC  AG  AT  CA  CC  CG  CT  GA  GC  GG  GT  TA
  # 0   0   0   0   0   0   0   0   0   0   0   0
  # TC  TG  TT
  # 0   0   0
  > dinuc <- colSums(dinuc)
  > dinuc[dinuc > 0]
  # Output:
  # [1] 34  70
  ```
Solution II

<table>
<thead>
<tr>
<th></th>
<th>GA</th>
<th>GC</th>
<th>GG</th>
<th>GT</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>42</td>
<td>51</td>
<td>102</td>
<td></td>
</tr>
</tbody>
</table>
Now, let's try get the alphabet frequency per every sequencing cycle.

This information is VERY useful to pick up errors!

Any ideas?
Solution

- Apropos is quite useful!
  
  ```r
  > apropos("alphabet")
  
  [1] "AA_ALPHABET"
  [2] "alphabet"
  [3] "alphabetByCycle"
  [4] "alphabetFrequency"
  [5] "alphabetScore"
  [6] "DNA_ALPHABET"
  [7] "RNA_ALPHABET"
  [8] ".__T__.alphabet:Biostrings"
  [9] ".__T__.alphabet:Biostrings"
  [10] ".__T__.alphabetByCycle:ShortRead"
  ```
Solution

[12] "_.T_.alphabetFrequency:Biostrings"
[13] "_.T_.alphabetScore:ShortRead"

▸ Lets use the function `alphabetByCycle`

> alph <- alphabetByCycle(sread(sread))
> dim(alph)

[1] 17 36

▸ Why did I use the `sread` accessor? Why does `alph` have 17 rows and 36 columns?

▸ Exercise: lets plot the alphabet by cycle relative frequency (only letters > 0) using lattice. Use only 1 panel and draw 1 line per alphabet letter present.

▸ Do you observe something unexpected?
Solution

```r
▶ vacio6

```library(lattice)```
```r
> alph2 <- as.data.frame(t(alph[rowSums(alph) > + 0, ]))
> head(alph2)
```

```
A  C  G  T
1  0  0 256  0
2  61 42  51 102
3  70 42  37 107
4  73 33  53  97
5  74 36  51  95
6  67 63  52  74
```
Solution

```r
> alph2 <- alph2/rowSums(alph2) * 100
> head(alph2)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00000</td>
<td>0.00000</td>
<td>100.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>2</td>
<td>23.82812</td>
<td>16.40625</td>
<td>19.92188</td>
<td>39.84375</td>
</tr>
<tr>
<td>3</td>
<td>27.34375</td>
<td>16.40625</td>
<td>14.45312</td>
<td>41.79688</td>
</tr>
<tr>
<td>4</td>
<td>28.51562</td>
<td>12.89062</td>
<td>20.70312</td>
<td>37.89062</td>
</tr>
<tr>
<td>5</td>
<td>28.90625</td>
<td>14.06250</td>
<td>19.92188</td>
<td>37.10938</td>
</tr>
<tr>
<td>6</td>
<td>26.17188</td>
<td>24.60938</td>
<td>20.31250</td>
<td>28.90625</td>
</tr>
</tbody>
</table>
```
Solution

```r
> print(xyplot(A + C + G + T ~ 1:nrow(alph2),
+       data = alph2, type = c("o",
+       "g"), auto.key = TRUE,
+       xlab = "Cycle", ylab = "Nucleotide Frequency")
```
Solution
qa report

- qa is a function that summarizes fastq files, export, etc and creates a series of summary plots.
- When working, it creates an html file.

```r
> args(qa)
function (dirPath, ...) NULL

> qa <- qa(sp)
> dir <- tempfile()

> report(qa, dest = dir)
> dir(paste(dir, "image", sep = "/"))
```

- Yet, we can still access some of the data through R:
qa report

> qa["baseCalls"]

    A   C   G   T   N
s_2_export.txt 9537 7480 7406 10537 40

Which file did qa use by default?
Filtering reads

* To end our cruise through SR, lets filter reads!

```r
> apropos("filter")
```

1. "alignDataFilter"
2. "alignQualityFilter"
3. ".__C__.FilterRules"
4. "chromosomeFilter"
5. ".__C__.SRFilter"
6. "dustyFilter"
7. "filter"
8. "Filter"
9. "Filter"
10. "filterBam"
11. "filterRules"
Filtering reads

[12] "filterRules<-
[13] "FilterRules"
[14] "idFilter"
[15] "nFilter"
[16] "occurrenceFilter"
[17] "polynFilter"
[18] "positionFilter"
[19] "srdistanceFilter"
[20] "srFilter"
[21] "strandFilter"
[22] ".__T__filterBam:Rsamtools"
[23] ".__T__Filter:base"
[24] ".__T__filterRules<-:IRanges"
[25] ".__T__filterRules:IRanges"
Filtering reads

[26] "._._srFilter:ShortRead"
[27] "uniqueFilter"

- The main class is srFilter, though many types are already coded.
  
  > nfilt <- nFilter()
  > cfilt <- chromosomeFilter("chr5.fa")
  > sfilt <- strandFilter("+")

- With the above filters we can now read in the reads from chromosome 5 in the plus strand.

- We can specify the filters when reading the file or to subset an AlignedRead object:
Filtering reads

```r
> chr5 <- readAligned(sp, "s_2_export.txt",
+   filter = cfilt)
> filt <- compose(cfilt, sfilt)
> chr5plus <- readAligned(sp, "s_2_export.txt",
+   filter = filt)
> length(chr5plus) == length(aln[filt(aln)])

[1] TRUE
```
Recap

- Universal format
- BAM is binary SAM
- SAM and BAM files can be ordered by the position of the reads (left to right on the genome)
- The model behind: read in pieces of files at a time.
- SAM format is doomed, any clues why?
Overview

- Similar to ShortRead
- Can read in pieces of files at a time
- In the near future: will be able to handle gaps!
Lets read a BAM file!

- After loading Rsamtools, next we need to construct a special object with `ScanBamParam`. Mainly this object specifies which parts of the chromosome / organism we want to read in and the columns of information we want.
- To do so we need to use some of the IRanges functionality. Don’t worry, we’ll cover it next week :) 
- Once we have the parameters, we can now read the BAM file using `scanBam`.
- `scanBam` can also read in files that are hosted on the web! :}
Lets read a BAM file!

```r
> library(Rsamtools)
> which <- RangesList(seq1 = IRanges(1000, 2000), seq2 = IRanges(c(100, 1000), c(1000, 2000)))
> which

SimpleRangesList of length 2
$seq1
IRanges of length 1
  start  end  width
[1] 1000 2000 1001

$seq2
IRanges of length 2
  start  end  width
```
Lets read a BAM file!

```
[1] 100 1000 901

> what <- c("rname", "strand", "pos", "qwidth", "seq")
> what

[1] "rname"  "strand"  "pos"    "qwidth"
[5] "seq"

> param <- ScanBamParam(which = which,
+                           what = what)
> param
```
Lets read a BAM file!

class: ScanBamParam
bamFlag: keep '0' bits: 2047; keep '1' bits: 2047
bamSimpleCigar: FALSE
bamReverseComplement: FALSE
bamTag:
bamWhich: 2 elements
bamWhat: rname, strand, pos, qwidth, seq

> bamFile <- system.file("extdata", "ex1.bam", package = "Rsamtools")
> bam <- scanBam(bamFile, param = param)
Exploring the output of scanBam

- The output is a list with a second list inside. At the lowest level we can find an object for each of the what columns we specified

```r
> class(bam)
[1] "list"

> names(bam)
[1] "seq1:1000-2000" "seq2:100-1000"

> lapply(bam, class)
```
Exploring the output of scanBam

```r
$`seq1:1000-2000`
[1] "list"

$`seq2:100-1000`
[1] "list"

$`seq2:1000-2000`
[1] "list"

> names(bam[[1]])
[1] "rname" "strand" "pos"   "qwidth"
[5] "seq"

> sapply(bam[[1]], class)
```

Exploring the output of scanBam

```
rname "factor" strand "factor"
pos "integer" qwidth "integer"
seq "DNASTringSet"
```
You might feel comfortable with such kind of data, though you might also like it in to view it in a tabular format such as a `DataFrame`:

```r
> lst <- lapply(names(bam[[1]]),
+   function(elt) {
+     do.call(c, unname(lapply(bam,
+                      "[[", elt)))
+   })
> names(lst) <- names(bam[[1]])
> head(do.call("DataFrame", lst))
```
### DataFrame

**DataFrame with 6 rows and 5 columns**

<table>
<thead>
<tr>
<th>rname</th>
<th>strand</th>
<th>pos</th>
<th>qwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;integer&gt;</td>
<td>&lt;integer&gt;</td>
<td>&lt;integer&gt;</td>
<td>&lt;integer&gt;</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

```bash
59 / 70
```
## DataFrame

<table>
<thead>
<tr>
<th></th>
<th>seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TATAGGAAATGCTTTACTGTCATAACTATGAAGA</td>
</tr>
<tr>
<td>2</td>
<td>ATTAGGAAATGCTTTACTGTCATAACTATGAAGAG</td>
</tr>
<tr>
<td>3</td>
<td>TTAGGAAATGCTTTACTGTCATAACTATGAAGAGA</td>
</tr>
<tr>
<td>4</td>
<td>TAGGAAATGCTTTACTGTCATAACTATGAAGAGAC</td>
</tr>
<tr>
<td>5</td>
<td>AGGAAATGCTTTACTGTCATAACTATGAAGAGACT</td>
</tr>
<tr>
<td>6</td>
<td>GGAAATGCTTTACTGTCATAACTATGAAGAGACTA</td>
</tr>
</tbody>
</table>

Note that it is a DataFrame and not a data.frame!
DataFrame

- We won’t use this kind of object much since we can also transform it into a GRanges object (next session!).
From the web!

- Just as an example, let's read in data from the web.
- We'll get data only from chromosome 6 bases 100k to 110k from the 1000 genomes project.
- If we wanted to download all the data, well, that's around 10GB! The output with scanBam is only around 2Mb in memory.

```r
> which <- RangesList(`6` = IRanges(100000L, + 110000L))
> param <- ScanBamParam(which = which)
> na19240bam <- scanBam(na19240url, + param = param)
```
From the web!

```r
> print(object.size(na19240bam),
+       units = "Mb")

2.1 Mb
```
Rsamtools has much more to offer

- We only took a quick glimpse at Rsamtools. It still has other useful functions if you are working with BAM files such as BamViews:

  ```
  use 'BamViews' to reference a set of disk-based BAM files to be processed (e.g., queried using `scanBam`) as a single experiment.
  ```

- There is also a function to read in gapped alignments:

  ```r
  > aln1_file <- system.file("extdata", 
  +     "ex1.bam", package = "Rsamtools")
  > aln1 <- readBamGappedAlignments(aln1_file)
  > head(aln1)
  ```
Rsamtools has much more to offer

GappedAlignments of length 6

<table>
<thead>
<tr>
<th>rname</th>
<th>strand</th>
<th>cigar</th>
<th>qwidth</th>
<th>start</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq1</td>
<td>+</td>
<td>36M</td>
<td>36</td>
<td>1</td>
<td>36</td>
</tr>
<tr>
<td>seq1</td>
<td>+</td>
<td>35M</td>
<td>35</td>
<td>3</td>
<td>37</td>
</tr>
<tr>
<td>seq1</td>
<td>+</td>
<td>35M</td>
<td>35</td>
<td>5</td>
<td>39</td>
</tr>
<tr>
<td>seq1</td>
<td>+</td>
<td>36M</td>
<td>36</td>
<td>6</td>
<td>41</td>
</tr>
<tr>
<td>seq1</td>
<td>+</td>
<td>35M</td>
<td>35</td>
<td>9</td>
<td>43</td>
</tr>
<tr>
<td>seq1</td>
<td>+</td>
<td>35M</td>
<td>35</td>
<td>13</td>
<td>47</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>width</th>
<th>ngap</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>0</td>
</tr>
<tr>
<td>35</td>
<td>0</td>
</tr>
<tr>
<td>35</td>
<td>0</td>
</tr>
<tr>
<td>36</td>
<td>0</td>
</tr>
</tbody>
</table>
Rsamtools has much more to offer

[5] 35 0
[6] 35 0

- And more to come as it’s been actively developed :)
Some practice

- From the `aln` object, extract the dinucleotide frequency for the last 2 cycles.
  1. Given the GC percentage of all cycles, did you expect the results you observe?
  2. Which is the read with NN at the end?
  3. Is there a significative difference vs the dinucleotide frequency of cycles 15 and 16?

- Load the `na19240url` object (note that `fpt` doesn’t work at `IBt`).
  1. Are all reads of the same length? If not, what is the distribution? Make a cumulative plot.
  2. Convert the `PhredQuality` instance to a quality matrix and make a plot of the median quality per cycle. Is there any trend in the quality?
Some practice

3. Make a third plot for the alphabet by cycle relative frequency (in percent). Do you observe anything unexpected?
Session Information

> sessionInfo()

R version 2.12.0 Under development (unstable) (2010-09-08 r52880)
Platform: x86_64-unknown-linux-gnu (64-bit)

locale:
[1] LC_CTYPE=en_US.utf8
[2] LC_NUMERIC=C
[3] LC_TIME=en_US.utf8
[4] LC_COLLATE=en_US.utf8
[5] LC_MONETARY=C
[6] LC_MESSAGES=en_US.utf8
[7] LC_PAPER=en_US.utf8
[8] LC_NAME=C
[9] LC_ADDRESS=C
[10] LC_TELEPHONE=C
[12] LC_IDENTIFICATION=C

attached base packages:
Session Information

[1] stats    graphics  grDevices
[4] utils    datasets  methods
[7] base

other attached packages:
[1] ShortRead_1.7.20
[2] Rsamtools_1.1.15
[3] lattice_0.19-11
[4] Biostrings_2.17.41
[5] GenomicRanges_1.1.25
[6] IRanges_1.7.34

loaded via a namespace (and not attached):
[1] Biobase_2.9.0  grid_2.12.0
[3] hwriter_1.2