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

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

September 29th, 2010

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

```
NAGAGGCCAGGCCATCTACCACCTTTTGTTGGAAATTTTGCTCTTTCAAC
+HWUSI-EAS636_0001:1:1:0:114#0/1
DOVYUQUYWSTWYYYYYYYYYVYYYYYYYYTPLSYYWWWRJRYYYYYWVTV
CGGAAGAGCGGTTCAGCAGGAATGCCGAGATCGGAAGAGCGGTTCAGCAT
+HWUSI-EAS636_0001:1:1:1:552#0/1
aaaaaaaK_Y^_b_ZaaaPIXa_VZWRNHZ^LHUHRRPUPVJIRQWQYXB
AGCGCATCTTGCGCTATGTGCAGCAGAGCGTGAGCCTTAACCTGATGCGC
```

- B

HTS data formats

```
HWUSI-EAS636    1       4       45      849     1900    0       1
GACTTAGGTCACTAAATACTTTAACCAATATAGGCA
abbabbbaabbabbababababaaaaaaaaabaa]`aa  ECK12.fasta             113
  F       36  146
HWUSI-EAS636    1       4       101     43      603     0       1
CTTAGGTCACTAAATACTTTAACCAATATAGGCATA
Z_b]J_^_Zaa]]^bbabba^_`aaabaa_^TT`_W      ECK12.fasta             115
      F       36  146
HWUSI-EAS636    1       4       109     1181    363     0       1
CTTAGGTCACTAAATACTTTAACCAATATAGGCATA
aa`baa`aaaabaaabaaaaa``aaaaa__``_aaa      ECK12.fasta             115
F       36  146
```

- C

HTS data formats

```
##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
```

HTS data formats

```
HWUSI-EAS636:8:120:1791:1350#0/1        -       gi|49175990|ref|
NC_000913.2|    1753520 TCGGACTGTAGAACTCTG      <:A8866?;8@:6><>?B
0       0:A>G,1:A>T,16:T>C
```

- E

```
HWUSI-EAS636_0009:8:120:16043:16103#0/1 +       gi|49175990|ref|
NC_000913.2|    4091324 GCCGAATTAGATGGC B############## 0
1:T>C,5:G>A,6:A>T
HWUSI-EAS636_0009:8:120:16043:16103#0/2 -       gi|49175990|ref|
NC_000913.2|    4091687 TTTTTGCTTCTTTTT ###########@23< 0
0:G>T,3:C>T,7:A>T
HWUSI-EAS636_0009:8:120:16061:14457#0/1 +       gi|49175990|ref|
NC_000913.2|    584209  GCCACCGAGTTAAAA C############## 0
11:C>A
HWUSI-EAS636_0009:8:120:16061:14457#0/2 -       gi|49175990|ref|
NC_000913.2|    584573  CTGAGAGTTGTACAT ############### 0
0:G>T,3:C>A,13:C>T
```

- F

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
CTAGTGGCTCATTGTAAATGTGTGGTTTAACTCGT <<<<<<<<<<0<<<<655<<7<<<:9<<3/:<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
AGTGGCTCATTGTAAATGTGTGGTTTAACTCGTCC <<<<<<<<<<<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
aaagttagagaagtttgacttctgtaggcaccatc -------------------------)))))))###
```

- I

  ```
  HWI-EAS88 3 2 1 451 945 CCAGAGCCCCCCGCTCACTCCTGAACCAGTCTCTC
  YQMIMIMMLMMIGIGMFICMFFFIMMHIIHAAGAH NM N
  HWI-EAS88 3 2 1 409 991 AGCCTCCCTCTTTCTGAATATACGGCAGAGCTGTT
  ZXZUYXZQYYXUZXYZYYZZXXZZIMFHXQSUPPO NM Y
  HWI-EAS88 3 2 1 451 939 ACCAAAAACACCACATACACGAGCAACACACGTAC
  LGDHLILLLLLLLIGFLLALDIFDILLHFIAECAE 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?

```
> library(ShortRead)
> exptPath <- system.file("extdata",
+     package = "ShortRead")
```

- For SR to recognize the path, we need to use SolexaPath:

```
> sp <- SolexaPath(exptPath)
> sp
```

Our first steps with ShortRead

```
class: SolexaPath
experimentPath: /usr/local/lib64/R/library/ShortRead/ex
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/
> 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.

```
> 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?

AlignedRead

- As the names imply, we can extract the positions with:

```
> 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:

```
> 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.
   0.00    0.00    0.00   17.04   37.00
   Max.
  72.00

> length(aln)

[1] 1000
```

AlignedRead

```
> 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

```
[5]        0
[6]        0
```

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:

```
> summary(position(aln))[7]

NA's
 594
```

Solution

- Lets take advantage of how R works by using vectors.

```
> 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:

```
> tenth <- head(position(aln), 10)
> tenth
```

Solution

```
 [1]       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]
```

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.
- Lets 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?

ShortReadQ

- In addition to AlignedRead, ShortReadQ objects completes the faimily of main objects in SR.

```
> sread
```

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

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

```
> 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))
```

ShortReadQ

```
class: SFastqQuality
quality:
  A BStringSet instance of length 6
    width seq
[1]    36 ]]]]]]]]]]]]Y]...]]]]VCHVMPLAS
[2]    36 ]]]]]]]]]]]]Y]...PV]T][PZPICCK
[3]    36 ]]]]Y]]]]]V]T]...]V]TMJEUXEFLA
[4]    36 ]]]]]]]]]]]]]]...]]]]RJRZTQLOA
[5]    36 ]]]]]]]]]]]]]]...]]]]]MJUJVLSS
[6]    36 ]]]]]]]]]]]Y]]...VO]W]VZMXVOLS
```

```
> length(sread)
```

```
[1] 256
```

```
> table(width(sread))
```

ShortReadQ

```
 36
256

> head(sread(sread))

  A DNAStringSet instance of length 6
    width seq
[1]    36 GGACTTTGTAGGAT...TTCCTTCTCCTGT
[2]    36 GATTTCTTACCTAT...AACAGCATCGGAC
[3]    36 GCGGTGGTCTATAG...TATCAATTTGGGT
[4]    36 GTTACCATGATGTT...TTTGGAGGTAAAA
[5]    36 GTATGTTTCTCCTG...TTCTTGAAGGCTT
[6]    36 GTTCTCTAAAAACC...CCCCTTCGGGGCG

> 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:

```
> `?`(BStringSet)
```

Solution I

- Lets use the sread, narrow, as.character and table functions:

```
> 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 I

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

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

Solution II

- While the above solution was *fine*, it did involve changing between types of objects.
- Lets use the dinucleotideFrequency function:

```
> dinuc <- dinucleotideFrequency(sread(narrow(sread,
+      start = 1, width = 2)))
> dinuc[1, ]
```

```
AA AC AG AT CA CC CG CT GA GC GG GT TA
 0  0  0  0  0  0  0  0  0  0  1  0  0
TC TG TT
 0  0  0
```

```
> dinuc <- colSums(dinuc)
> dinuc[dinuc > 0]
```

Solution II

```
 GA  GC  GG  GT
 61  42  51 102
```

Alphabet Frequency

- Now, lets 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!

```
> 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"
[11] ".__T__alphabetFrequency:Biostrings"
```

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

- vacio6

```
> library(lattice)
> 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
```

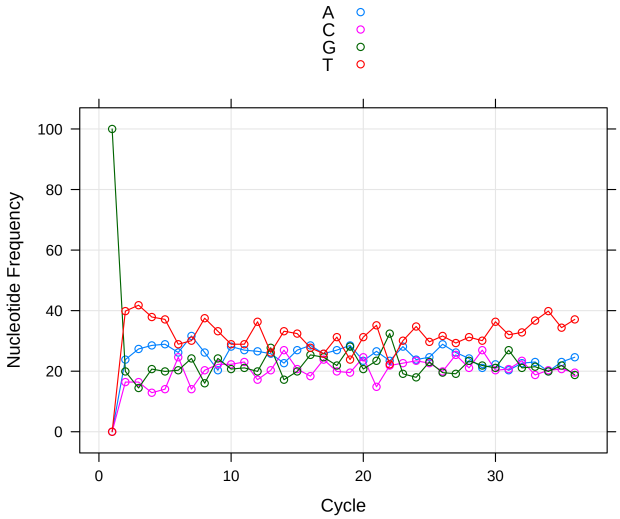
Solution

```
> alph2 <- alph2/rowSums(alph2) *
+     100
> head(alph2)
         A        C         G        T
1  0.00000  0.00000 100.00000  0.00000
2 23.82812 16.40625  19.92188 39.84375
3 27.34375 16.40625  14.45312 41.79688
4 28.51562 12.89062  20.70312 37.89062
5 28.90625 14.06250  19.92188 37.10938
6 26.17188 24.60938  20.31250 28.90625
```

Solution

```
> 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.

```
> 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!

```
> 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] ".__T__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 chromsome 5 in the plus strand.
- We can specify the filters when reading the file or to subset an AlignedRead object:

Filtering reads

```
> 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.
- Definition: `http://samtools.sourceforge.net/SAM1.pdf`
- Related tools: `http://samtools.sourceforge.net/`
- 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 chromsome / organism we want ot 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 paramters, 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!

```
> 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
[2]  1000 2000  1001
```

```
> 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

```
> class(bam)
```

```
[1] "list"
```

```
> names(bam)
```

```
[1] "seq1:1000-2000" "seq2:100-1000"
[3] "seq2:1000-2000"
```

```
> lapply(bam, class)
```

Exploring the output of scanBam

```
$`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          strand
       "factor"        "factor"
            pos          qwidth
      "integer"       "integer"
            seq
"DNAStringSet"
```

DataFrame

- 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:

```
> 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
      rname    strand       pos
  <integer> <integer> <integer>
1         1         1       970
2         1         1       971
3         1         1       972
4         1         1       973
5         1         1       974
6         1         2       975
     qwidth
  <integer>
1        35
2        35
3        35
```

DataFrame

```
4          35
5          35
6          35
                                    seq
                         <DNAStringSet>
1 TATTAGGAAATGCTTTACTGTCATAACTATGAAGA
2 ATTAGGAAATGCTTTACTGTCATAACTATGAAGAG
3 TTAGGAAATGCTTTACTGTCATAACTATGAAGAGA
4 TAGGAAATGCTTTACTGTCATAACTATGAAGAGAC
5 AGGAAATGCTTTACTGTCATAACTATGAAGAGACT
6 GGAAATGCTTTACTGTCATAACTATGAAGAGACTA
```

- 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, lets 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 donwload all the data, well, that's around 10GB! The output with scanBam is only around 2Mb in memory.

```
> which <- RangesList(`6` = IRanges(100000L,
+     110000L))
> param <- ScanBamParam(which = which)
> na19240url <- "ftp://ftp-trace.ncbi.nih.gov/1000genom
> na19240bam <- scanBam(na19240url,
+     param = param)
```

From the web!

```
> 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 âĂŸscanBamâĂŹ) as a single âĂŸexperimentâĂŹ.
- There is also a function to read in gapped alignments:

```
> 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
    rname strand cigar qwidth start end
[1]  seq1      +   36M     36     1  36
[2]  seq1      +   35M     35     3  37
[3]  seq1      +   35M     35     5  39
[4]  seq1      +   36M     36     6  41
[5]  seq1      +   35M     35     9  43
[6]  seq1      +   35M     35    13  47
    width ngap
[1]    36    0
[2]    35    0
[3]    35    0
[4]    36    0
```

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 wort 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
[11] LC_MEASUREMENT=en_US.utf8
[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
```