An Introduction to *ShortRead*

Martin Morgan

Modified: 21 October, 2013. Compiled: October 13, 2014

```
> library("ShortRead")
```

The *ShortRead* package provides functionality for working with FASTQ files from high throughput sequence analysis. The package also contains functions for legacy (single-end, ungapped) aligned reads; for working with BAM files, please see the *Rsamtools*, *GenomicRanges*, and related packages.

1 Sample data

Sample FASTQ data are derived from ArrayExpress record E-MTAB-1147. Paired-end FASTQ files were retrieved and then sampled to 20,000 records with

```
> sampler <- FastqSampler('E-MTAB-1147/fastq/ERR127302_1.fastq.gz', 20000)
> set.seed(123); ERR127302_1 <- yield(sampler)
> sampler <- FastqSampler('E-MTAB-1147/fastq/ERR127302_2.fastq.gz', 20000)
> set.seed(123); ERR127302_2 <- yield(sampler)
```

2 Functionality

Functionality is summarized in Table 1.

Input FASTQ files are large so processing involves iteration in 'chunks' using `FastqStreamer`

```
> strm <- FastqStreamer("a.fastq.gz")
> repeat {
+     fq <- yield(strm)
+     if (length(fq) == 0)
+         break
+     ## process chunk
+ }
```

or drawing a random sample from the file

```
> sampler <- FastqSampler("a.fastq.gz")
> fq <- yield(sampler)
```

The default size for both streams and samples is 1M records; this volume of data fits easily into memory. Small FASTQ files can be read in to memory in their entirety using `readFastq`; we do this for our sample data

```
> fl <- system.file(package="ShortRead", "extdata", "E-MTAB-1147",
+                   "ERR127302_1_subset.fastq.gz")
> fq <- readFastq(fl)
```

Input	
`FastqStreamer`	Iterate through FASTQ files in chunks
`FastqSampler`	Draw random samples from FASTQ files
`readFastq`	Read an entire FASTQ file into memory
`writeFastq`	Write FASTQ objects to a connection (file)
Sequence and quality summary	
`alphabetFrequency`	Nucleotide or quality score use per read
`alphabetByCycle`	Nucleotide or quality score use by cycle
`alphabetScore`	Whole-read quality summary
`encoding`	Character / 'phred' score mapping
Quality assessment	
`qa`	Visit FASTQ files to collect QA statistics
`report`	Generate a quality assessment report
Filtering and trimming	
`srFilter`	Pre-defined and bespoke filters
`trimTails`, etc.	Trim low-quality nucleotides
`narrow`	Remove leading / trailing nucleotides
`tables`	Summarize read occurrence
`srduplicated`, etc.	Identify duplicate reads
`filterFastq`	Filter reads from one file to another

Table 1: Key functions for working with FASTQ files

DNAStringSet	(*Biostrings*) Short read sequences
FastqQuality, etc.	Quality encodings
ShortReadQ	Reads, quality scores, and ids

Table 2: Primary data types in the *ShortRead* package

The result of data input is an instance of class *ShortReadQ* (Table 2). This class contains reads, their quality scores, and optionally the id of the read.

```
> fq
```

```
class: ShortReadQ
length: 20000 reads; width: 72 cycles
```

```
> fq[1:5]
```

```
class: ShortReadQ
length: 5 reads; width: 72 cycles
```

```
> head(sread(fq), 3)
```

```
  A DNAStringSet instance of length 3
    width seq
[1]    72 GTCTGCTGTATCTGTGTCGGCTGTCTCGCGGGACATGAAGTCAATGAAGGCCTGGAATGTCACTACCCCCAG
[2]    72 CTAGGGCAATCTTTGCAGCAATGAATGCCAATGGGTAGCCAGTGGCTTTTGAGGCCAGAGCAGACCTTCGGG
[3]    72 TGGGCTGTTCCTTCTCACTGTGGCCTGACTAAAACCCAGTGGCATTAAGAAAGAGTCACGTTTCCCAAGTCT
```

```
> head(quality(fq), 3)
```

```
class: FastqQuality
quality:
  A BStringSet instance of length 3
    width seq
[1]    72 HHHHHHHHHHHHHHHHHHHHHEBDBB?B:BBGG<DDAA?AABFEFBDBD@DDECEE3>:?;@@@>?=BAB?##
```

```
[2]     72 IIIIHIIIGIIIIIIIHIIIIEGBGHIIIIHGIIHIIIIIIIHIIIHIIIIIGIIIEGIIGBGE@DDGGGIG
[3]     72 GGHBHGBGGGHHHHDHHHHHHHHHHFGHHHHHHHHHHHHHHHHHHHHHHHGHFHHHHHHHHHHHHHH8AGDGGG>
```

The reads are represented as *DNAStringSet* instances, and can be manipulated with the rich tools defined in the *Biostrings* package. The quality scores are represented by a class that represents the quality encoding inferred from the file; the encoding in use can be discovered with

```
> encoding(quality(fq))
```

```
 !  "  #  $  %  &  '  (  )  *  +  ,  -  .  /  0  1  2  3  4  5  6  7  8  9  :  ;  <  =  >
 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
 ?  @  A  B  C  D  E  F  G  H  I  J
30 31 32 33 34 35 36 37 38 39 40 41
```

The primary source of documentation for these classes is `?ShortReadQ` and `?QualityScore`.

3 Common work flows

3.1 Quality assessment

FASTQ files are often used for basic quality assessment, often to augment the purely technical QA that might be provided by the sequencing center with QA relevant to overall experimental design. A QA report is generated by creating a vector of paths to FASTQ files

```
> fls <- dir("/path/to", "*fastq$", full=TRUE)
```

collecting statistics over the files

```
> qaSummary <- qa(fls, type="fastq")
```

and creating and viewing a report

```
> browseURL(report(qaSummary))
```

By default, the report is based on a sample of 1M reads.

These QA facilities are easily augmented by writing custom functions for reads sampled from files, or by explorting the elements of the object returned from `qa()`, e.g., for an analysis of ArrayExpress experiment E-MTAB-1147:

```
> qaSummary
```

```
class: FastqQA(10)
QA elements (access with qa[["elt"]]):
  readCounts: data.frame(16 3)
  baseCalls: data.frame(16 5)
  readQualityScore: data.frame(8192 4)
  baseQuality: data.frame(1504 3)
  alignQuality: data.frame(16 3)
  frequentSequences: data.frame(800 4)
  sequenceDistribution: data.frame(1953 4)
  perCycle: list(2)
    baseCall: data.frame(5681 4)
    quality: data.frame(44246 5)
  perTile: list(2)
    readCounts: data.frame(0 4)
    medianReadQualityScore: data.frame(0 4)
  adapterContamination: data.frame(16 1)
```

For instance, the count of reads in each lane is summarized in the `readCounts` element, and can be displayed as

```
> head(qaSummary[["readCounts"]])
```

```
                         read filter aligned
ERR127302_1.fastq.gz 29741852     NA      NA
ERR127302_2.fastq.gz 29741852     NA      NA
ERR127303_1.fastq.gz 32665567     NA      NA
ERR127303_2.fastq.gz 32665567     NA      NA
ERR127304_1.fastq.gz 31876181     NA      NA
ERR127304_2.fastq.gz 31876181     NA      NA
```

```
> head(qaSummary[["baseCalls"]])
```

```
                             A        C        G        T     N
ERR127302_1.fastq.gz 16439860 19641395 19547421 16335620 35704
ERR127302_2.fastq.gz 16238041 20020655 19608896 16060661 71747
ERR127303_1.fastq.gz 16826258 19204659 19448727 16507994 12362
ERR127303_2.fastq.gz 16426991 19822132 19374419 16324978 51480
ERR127304_1.fastq.gz 16279217 19740457 19879137 16089405 11784
ERR127304_2.fastq.gz 15984998 20297064 19812474 15853510 51954
```

The `readCounts` element contains a data frame with 1 row and 3 columns (these dimensions are indicated in the parenthetical annotation of `readCounts` in the output of `qaSummary`). The rows represent different lanes. The columns indicated the number of reads, the number of reads surviving the Solexa filtering criteria, and the number of reads aligned to the reference genome for the lane. The `baseCalls` element summarizes base calls in the unfiltered reads.

The functions that produce the report tables and graphics are internal to the package. They can be accessed through calling ShortRead:::functionName where functionName is one of the functions listed below, organized by report section.

Run Summary : .ppnCount, .df2a, .laneLbl, .plotReadQuality
Read Distribution : .plotReadOccurrences, .freqSequences
Cycle Specific : .plotCycleBaseCall, .plotCycleQuality
Tile Performance : .atQuantile, .colorkeyNames, .plotTileLocalCoords, .tileGeometry, .plotTileCounts, .plotTileQualityScore
Alignment : .plotAlignQuality
Multiple Alignment : .plotMultipleAlignmentCount
Depth of Coverage : .plotDepthOfCoverage
Adapter Contamination : .ppnCount

3.2 Filtering and trimming

It is straight-forward to create filters to eliminate reads or to trim reads based on diverse characteristics. The basic structure is to open a FASTQ file, iterate through chunks of the file performing filtering or trimming steps, and appending the filtered data to a new file.

```
> myFilterAndTrim <-
+     function(fl, destination=sprintf("%s_subset", fl))
+ {
+     ## open input stream
+     stream <- open(FastqStreamer(fl))
+     on.exit(close(stream))
+
+     repeat {
+         ## input chunk
+         fq <- yield(stream)
+         if (length(fq) == 0)
+             break
+
```

```
+         ## trim and filter, e.g., reads cannot contain 'N'...
+         fq <- fq[nFilter()(fq)]  # see ?srFilter for pre-defined filters
+         ## trim as soon as 2 of 5 nucleotides has quality encoding less
+         ## than "4" (phred score 20)
+         fq <- trimTailw(fq, 2, "4", 2)
+         ## drop reads that are less than 36nt
+         fq <- fq[width(fq) < 36]
+
+         ## append to destination
+         writeFastq(fq, destination, "a")
+     }
+ }
```

This is memory efficient and flexible. Care must be taken to coordinate pairs of FASTQ files representing paired-end reads, to preserve order.

4 Using ShortRead for data exploration

4.1 Data I/O

ShortRead provides a variety of methods to read data into *R*, in addition to `readAligned`.

4.1.1 `readXStringColumns`

`readXStringColumns` reads a column of DNA or other sequence-like data. For instance, the Solexa files `s_N_export.txt` contain lines with the following information:

```
> ## location of file
> exptPath <- system.file("extdata", package="ShortRead")
> sp <- SolexaPath(exptPath)
> pattern <- "s_2_export.txt"
> fl <- file.path(analysisPath(sp), pattern)
> strsplit(readLines(fl, n=1), "\t")
```

```
[[1]]
 [1] "HWI-EAS88"                          "3"
 [3] "2"                                  "1"
 [5] "451"                                "945"
 [7] ""                                   ""
 [9] "CCAGAGCCCCCCGCTCACTCCTGAACCAGTCTCTC" "YQMIMIMMLMMIGIGMFICMFFFIMMHIIHAAGAH"
[11] "NM"                                 ""
[13] ""                                   ""
[15] ""                                   ""
[17] ""                                   ""
[19] ""                                   ""
[21] ""                                   "N"
```

```
> length(readLines(fl))
```

```
[1] 1000
```

Column 9 is the read, and column 10 the ASCII-encoded Solexa Fastq quality score; there are 1000 lines (i.e., 1000 reads) in this sample file.

Suppose the task is to read column 9 as a *DNAStringSet* and column 10 as a *BStringSet*. *DNAStringSet* is a class that contains IUPAC-encoded DNA strings (IUPAC code allows for nucleotide ambiguity); *BStringSet* is a class that contains any character with ASCII code 0 through 255. Both of these classes are defined in the *Biostrings* package. `readXStringColumns` allows us to read in columns of text as these classes.

Important arguments for `readXStringColumns` are the `dirPath` in which to look for files, the `pattern` of files to parse, and the `colClasses` of the columns to be parsed. The `dirPath` and `pattern` arguments are like `list.files`. `colClasses` is like the corresponding argument to `read.table`: it is a *list* specifying the class of each column to be read, or `NULL` if the column is to be ignored. In our case there are 21 columns, and we would like to read in columns 9 and 10. Hence

```
> colClasses <- rep(list(NULL), 21)
> colClasses[9:10] <- c("DNAString", "BString")
> names(colClasses)[9:10] <- c("read", "quality")
```

We use the class of the type of sequence (e.g., *DNAString* or *BString*), rather than the class of the set that we will create (e.g., *DNAStringSet* or *BStringSet*). Applying names to `colClasses` is not required, but makes subsequent manipulation easier. We are now ready to read our file

```
> cols <- readXStringColumns(analysisPath(sp), pattern, colClasses)
> cols
```

```
$read
  A DNAStringSet instance of length 1000
      width seq
   [1]    35 CCAGAGCCCCCCGCTCACTCCTGAACCAGTCTCTC
   [2]    35 AGCCTCCCTCTTTCTGAATATACGGCAGAGCTGTT
   [3]    35 ACCAAAAACACCACATACACGAGCAACACACGTAC
   [4]    35 AATCGGAAGAGCTCGTATGCCGGCTTCTGCTTGGA
   [5]    35 AAAGATAAACTCTAGGCCACCTCCTCCTTCTTCTA
   ...   ... ...
 [996]    35 GTGGCAGCGGTGAGGCGGCGGGGGGGGGTTGTTTG
 [997]    35 GTCGGAGGTCAGCAAGCTGTAGTCGGTGTAAAGCT
 [998]    35 GTCATAAATTGGACAGTGTGGCTCCAGTATTCTCA
 [999]    35 ATCTACATTAAGGTCAATTACAATGATAAATAAAA
[1000]    35 TTCTCAGCCATTCAGTATTCCTCAGGTGAAAATTC

$quality
  A BStringSet instance of length 1000
      width seq
   [1]    35 YQMIMIMMLMMIGIGMFICMFFFIMMHIIHAAGAH
   [2]    35 ZXZUYXZQYYXUZXYZYYZZXXZZIMFHXQSUPPO
   [3]    35 LGDHLILLLLLLLIGFLLALDIFDILLHFIAECAE
   [4]    35 JJYYIYVSYYYYYYYYSDYYWVUYYNNVSVQQELQ
   [5]    35 LLLILIIIDLLHLLLLLLLLLLALLLLHLLLLEL
   ...   ... ...
 [996]    35 ZZZZZZZYZZYUYZYUYZKYUDUZIYYODJGUGAA
 [997]    35 ZZZZZZZZZZZZZZZZZZYZZYXXZYSSXXUUHHQ
 [998]    35 ZZZZZZZZZZZZZZZZYZZZZYZZZZYZZXZUUUUS
 [999]    35 ZZZZZZZZZZZZYXZYZYZZYZYZZXKZSYXUUNUN
[1000]    35 ZZZZZZZZZZZZZZZYZZZZZZZZYYSYSZXUUUUU
```

The file has been parsed, and appropriate data objects were created.

A feature of `readXStringColumns` and other input functions in the *ShortRead* package is that all files matching `pattern` in the specified `dirPath` will be read into a single object. This provides a convenient way to, for instance, parse all tiles in a Solexa lane into a single *DNAStringSet* object.

There are several advantages to reading columns as *XStringSet* objects. These are more compact than the corresponding character representation:

```
> object.size(cols$read)
```

```
50840 bytes
```

```
> object.size(as.character(cols$read))
```

```
94280 bytes
```

They are also created much more quickly. And the *DNAStringSet* and related classes are used extensively in *ShortRead*, *Biostrings*, *BSgenome* and other packages relevant to short read technology.

4.2 Sorting

Short reads can be sorted using `srsort`, or the permutation required to bring the short read into lexicographic order can be determined using `srorder`. These functions are different from `sort` and `order` because the result is independent of the locale, and they operate quickly on *DNAStringSet* and *BStringSet* objects.

The function `srduplicated` identifies duplicate reads. This function returns a logical vector, similar to `duplicated`. The negation of the result from `srduplicated` is useful to create a collection of unique reads. An experimental scenario where this might be useful is when the sample preparation involved PCR. In this case, replicate reads may be due to artifacts of sample preparation, rather than differential representation of sequence in the sample prior to PCR.

4.3 Summarizing read occurrence

The `tables` function summarizes read occurrences, for instance,

```
> tbls <- tables(fq)
> names(tbls)
```

```
[1] "top"          "distribution"
```

```
> tbls$top[1:5]
```

```
CTATTCTCTACAAACCACAAAGACATTGGAACACTATACCTATTATTCGGCGCATGAGCTGGAGTCCTAGGC
                                                                      7
GTTTGGTCTAGGGTGTAGCCTGAGAATAGGGGAAATCAGTGAATGAAGCCTCCTATGATGGCAAATACAGCT
                                                                      7
CGATAACGTTGTAGATGTGGTCGTTACCTAGAAGGTTGCCTGGCTGGCCCAGCTCGGCTCGAATAAGGAGGC
                                                                      6
CTAGCATTTACCATCTCACTTCTAGGAATACTAGTATATCGCTCACACCTCATATCCTCCCTACTATGCCTA
                                                                      6
CACGAGCATATTTCACCTCCGCTACCATAATCATCGCTATCCCCACCGGCGTCAAAGTATTTAGCTGACTCG
                                                                      5
```

```
> head(tbls$distribution)
```

```
  nOccurrences nReads
1            1  19291
2            2    247
3            3     34
4            4     18
5            5      3
6            6      2
```

The `top` component returned by `tables` is a list tallying the most commonly occurring sequences in the short reads. Knowledgeable readers will recognize the top-occurring read as a close match to one of the manufacturer adapters.

The `distribution` component returned by `tables` is a data frame that summarizes how many reads (e.g., 19291) are represented exactly 1 times.

4.4 Finding near matches to short sequences

Facilities exist for finding reads that are near matches to specific sequences, e.g., manufacturer adapter or primer sequences. `srdistance` reports the edit distance between each read and a reference sequence. `srdistance` is implemented to work efficiently for reference sequences whose length is of the same order as the reads themselves (10's to 100's of bases). To find reads close to the most common read in the example above, one might say

```
> dist <- srdistance(sread(fq), names(tbls$top)[1])[[1]]
> table(dist)[1:10]
```

```
dist
 0  4  6 10 14 18 20 21 31 32
 7  1  3  1  3  1  4  1  3 11
```

'Near' matches can be filtered, e.g.,

```
> fqSubset <- fq[dist>4]
```

A different strategy can be used to tally or eliminate reads that consist predominantly of a single nucleotide. `alphabetFrequency` calculates the frequency of each nucleotide (in DNA strings) or letter (for other string sets) in each read. Thus one could identify and eliminate reads with more than 30 adenine nucleotides with

```
> countA <- alphabetFrequency(sread(fq))[,"A"]
> fqNoPolyA <- fq[countA < 30]
```

`alphabetFrequency`, which simply counts nucleotides, is much faster than `srdistance`, which performs full pairwise alignment of each read to the subject.

Users wanting to use *R* for whole-genome alignments or more flexible pairwise aligment are encouraged to investigate the *Biostrings* package, especially the *PDict* class and `matchPDict` and `pairwiseAlignment` functions.

5 Legacy support for early file formats

The *ShortRead* package contains functions and classes to support early file formats and ungapped alignments. Help pages are flagged as 'legacy'; versions of *ShortRead* prior to 1.21 (*Bioconductor* version 2.13) contain a vignette illustrating common work flows with these file formats.

6 sessionInfo

```
> toLatex(sessionInfo())
```

- R version 3.1.1 Patched (2014-09-25 r66681), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: BiocGenerics 0.12.0, BiocParallel 1.0.0, Biostrings 2.34.0, GenomeInfoDb 1.2.0, GenomicAlignments 1.2.0, GenomicRanges 1.18.0, IRanges 2.0.0, Rsamtools 1.18.0, S4Vectors 0.4.0, ShortRead 1.24.0, XVector 0.6.0

- Loaded via a namespace (and not attached): BBmisc 1.7, BatchJobs 1.4, Biobase 2.26.0, BiocStyle 1.4.0, DBI 0.3.1, RColorBrewer 1.0-5, RSQLite 0.11.4, base64enc 0.1-2, bitops 1.0-6, brew 1.0-6, checkmate 1.4, codetools 0.2-9, digest 0.6.4, fail 1.2, foreach 1.4.2, grid 3.1.1, hwriter 1.3.2, iterators 1.0.7, lattice 0.20-29, latticeExtra 0.6-26, sendmailR 1.2-1, stringr 0.6.2, tools 3.1.1, zlibbioc 1.12.0