BCGES short courses, session 1
Introduction

The aim of this first session is to introduce the tools that we will use to manipulate sequence data. Generally speaking, we will deal with either `R`, bash scripting or Galaxy.

`R` is not usually the preferred way to deal with HTS data. One reason for this is that the general approach of `R` is to load data into the RAM as a first instance, which is not practical when the datasets are very large. Shell scripts (aka command line tools) that read the data on a line per line basis are usually preferred. Nevertheless, `R` has developed tools to overcome these limitations and there is, in fact, surprisingly much that one can do with it. It is also a practical tools for teaching purposes, which is useful in the context of these short courses. So we will use `R` mostly to play with the data and show the sort of things one may want to do with it.

Contents

1 Basic fastq reading and quality control

1.1 Basic shell scripting to read NGS files

Many of the standard HTS formats are simple text files, with tightly defined specifications to allow effective parsing. Some of these files can be compressed and/or indexed to enable quicker access. The first esstential step is to be confortable with reading/compressing/uncompressing various text files.

```
head ../../data/fastq_files/fastq1_1.txt

## @A81CH8ABXX:4:1101:1524:1813#NGACCAAT/1
## NACCACTCAGCTCTGGCCAATTATTGCCGTGCAGGAGTGTGGGCTCCTAGTGGCAGGGGGTCTGGAACTGTGGAAGAAGCAGGCAAACGC
## +
## BQXXQY[V[Yccc_c__V\_ccc___\\[X^^^^^BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
## @A81CH8ABXX:4:1101:1583:1836#NGACCAAT/1
## NTCCCCAAAGCACAGGGCTCAGCTCCAGAGGGAGACGGGCTGGGCTGTCAGCGGGCCCAGGGGCACGCCACTGTTCAGAACAACTGGTTG
## +
## BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
## @A81CH8ABXX:4:1101:1729:1852#TGACCAAT/1
## NCTATGGACTGTGGTAAAGCTAGGATTAGTAACCAGACATTACTTACCTTGGCTCCGATCTGGTTGCCACACTGGCCAATCTGAATATGG
```

```
tail ../../data/fastq_files/fastq1_1.txt

## +
## ggggggggggggggggggggggggggggggggggggggggggggggggegfgggggaeeede[^d[_Y``b]a[bZXZ[Y`U^BBBBBBB
## @A81CH8ABXX:4:1101:18871:2349#TGACCAAT/1
## TCTGTGCCGATCCCAGAGTGCCCTGGGTGAAGATGATTCTCAATAAGCTGAGCCAATGAAGAGCCTACTCTGATGACCGTGGCCTTGGCT
## +
## eedeee]decdbdbacca`bdbbdac_adWdb__dcee\ceeeeedeeee_ede^ddbdaUWbbd__aa[dee`e\bb_cad`dZcZc\b
## @A81CH8ABXX:4:1101:18909:2349#TGACCAAT/1
## TGGTCACTAGCTCTGGATGGTGCTCCAACAATGGTCACTTCTCGCGGAGAACACAAACACCAGCATCACAGCGCTGGGTTCCCATGGATG
## +
## gggggfgggggggggggdfgdfgggggggffggggdggggggfgfggfbggggggggggfgegggcgg^gggeffedfgggfeggeeg
```

If you need more information, shell scripts have useful manual page that can be accessed using the `man` command.

```
man head
man tail
```

One can parse a text file using a variety of commands, and it is useful to become familiar with the following:

```
cat ../../data/fastq_files/fastq1_1.txt
less ../../data/fastq_files/fastq1_1.txt
more ../../data/fastq_files/fastq1_1.txt
less -S ../../data/fastq_files/fastq1_1.txt
```

Exercise: Can you see the differences between these various ways of reading a file?

A routine that is often useful is `wc` that counts the words/character/lines of a file. Try:

```
wc  ../../data/fastq_files/fastq1_1.txt
wc  -l ../../data/fastq_files/fastq1_1.txt
```

Exercise: Using the man page, find a way to print the first 20 lines of a fastq file (and what about the last 20 lines)?
Exercise: Based on the `wc` output, how many reads do these fastq files contain?

1.2 Using the shortRead package in R

We start by loading one of the most relevant library, called “ShortReads”. This package may not be installed but it can easily done so by running:

```
source("http://bioconductor.org/biocLite.R")
biocLite("ShortRead")
```

```
library('ShortRead')

## Loading required package:  methods
## Loading required package:  BiocGenerics
## Loading required package:  parallel
##
## Attaching package:  'BiocGenerics'
##
## The following objects are masked from 'package:parallel':
##
##    clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##    clusterExport, clusterMap, parApply, parCapply, parLapply,
##    parLapplyLB, parRapply, parSapply, parSapplyLB
##
## The following object is masked from 'package:stats':
##
##    xtabs
##
## The following objects are masked from 'package:base':
##
##    anyDuplicated, append, as.data.frame, as.vector, cbind,
##    colnames, duplicated, eval, evalq, Filter, Find, get,
##    intersect, is.unsorted, lapply, Map, mapply, match, mget,
##    order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,
##    rbind, Reduce, rep.int, rownames, sapply, setdiff, sort,
##    table, tapply, union, unique, unlist
##
## Loading required package:  IRanges
## Loading required package:  GenomicRanges
## Loading required package:  XVector
## Loading required package:  Biostrings
## Loading required package:  lattice
## Loading required package:  Rsamtools
```

As a starting point it is possible to read some of the examples fastq files and create relevant `R` objects.

```
fastq1.1 <- readFastq('../../data/fastq_files/fastq1_1.txt')
fastq1.2 <- readFastq('../../data/fastq_files/fastq1_2.txt')
```

We can now display the sequences and the qualities. Note that specific classes have been defined to store each of these objects. Much work has gone into figuring out how to do this.

```
reads <- sread(fastq1.1)
class(reads)

## [1] "DNAStringSet"
## attr(,"package")
## [1] "Biostrings"

head(as.character(reads))

## [1] "NACCACTCAGCTCTGGCCAATTATTGCCGTGCAGGAGTGTGGGCTCCTAGTGGCAGGGGGTCTGGAACTGTGGAAGAAGCAGGCAAACGC"
## [2] "NTCCCCAAAGCACAGGGCTCAGCTCCAGAGGGAGACGGGCTGGGCTGTCAGCGGGCCCAGGGGCACGCCACTGTTCAGAACAACTGGTTG"
## [3] "NCTATGGACTGTGGTAAAGCTAGGATTAGTAACCAGACATTACTTACCTTGGCTCCGATCTGGTTGCCACACTGGCCAATCTGAATATGG"
## [4] "NACTGAAAAAGGATGCTTTGGAAAAAGAAAGTGGGTCTGGCAACACTGACTCAACCTTGAATTCCCCGCACGATGACACGGATGACAGGG"
## [5] "GTTATTTAAGCCACCCAGTCTGTGTTTGTTATGGCAGGCTGAGGAGACTATGACAGGAACCAAACACAAAAAAAACCAAAACTCTGGAGG"
## [6] "ATTTTGTAAACCTCTTATCCTTAGATCCAAAAGATATGTTCATCTAGGCTTGATAAGCACATGTGCATTTATACCACACTCTATAGTTCT"

ids <- id(fastq1.1)
class(ids)

## [1] "BStringSet"
## attr(,"package")
## [1] "Biostrings"

head(as.character(ids))

## [1] "A81CH8ABXX:4:1101:1524:1813#NGACCAAT/1"
## [2] "A81CH8ABXX:4:1101:1583:1836#NGACCAAT/1"
## [3] "A81CH8ABXX:4:1101:1729:1852#TGACCAAT/1"
## [4] "A81CH8ABXX:4:1101:1642:1867#TGACCAAT/1"
## [5] "A81CH8ABXX:4:1101:1715:1891#TGACCAAT/1"
## [6] "A81CH8ABXX:4:1101:1624:1941#TGACCAAT/1"
```

1.3 Using the Galaxy server

Start by identifying a working instance of the Galaxy server from this location. There are multiple choices here, so feel free to experiment. In case of doubt, I used the following server: http://biominavm-galaxy.biomina.be/galaxy/.
Exercise: Create an account on one of these Galaxy servers and upload the pair of fastq *fastq2_1.txt* and *fastq2_2.txt*.

2 Reading and interpreting quality scores

The quality scores can be read directly at the command line level, using commands like `head` or `tail`. But it is also possible and even practical to use R to do this. For example, following up on the example above:

```
quals <- quality(fastq1.1)
class(quals)

## [1] "SFastqQuality"
## attr(,"package")
## [1] "ShortRead"

quals

## class: SFastqQuality
## quality:
##   A BStringSet instance of length 2500
##        width seq
##    [1]    90 BQXXQY[V[Yccc_c__V\_ccc___\\[X...BBBBBBBBBBBBBBBBBBBBBBBBBBBBB
##    [2]    90 BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB...BBBBBBBBBBBBBBBBBBBBBBBBBBBBB
##    [3]    90 BWSSQVUVUTTQVSVUUUWW___BBBBBBB...BBBBBBBBBBBBBBBBBBBBBBBBBBBBB
##    [4]    90 BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB...BBBBBBBBBBBBBBBBBBBBBBBBBBBBB
##    [5]    90 gefgggggdgegfgggdfegee`eedddbd...BBBBBBBBBBBBBBBBBBBBBBBBBBBBB
##    ...   ... ...
## [2496]    90 gfgggggggegggggggfggefefggfcga...edeedOZ^`IZXYW^\YW]`ccX_adbdb
## [2497]    90 ggggggggggggggfggggggggdgfgcgg...ecgggegdfeeeg^dbdee`bNddeee^a
## [2498]    90 gggggggggggggggggggggggggggggg...e[^d[_Y``b]a[bZXZ[Y`U^BBBBBBB
## [2499]    90 eedeee]decdbdbacca`bdbbdac_adW...Wbbd__aa[dee`e\bb_cad`dZcZc\b
## [2500]    90 gggggfgggggggggggdfgdgfggggggg...fgegggcgg^gggeffedfgggfeggeeg
```

The `ShortRead` package will attempt to guess what these quality scores mean, for example see:

```
encoding(quality(fastq1.1))

##  ;  <  =  >  ?  @  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P  Q  R  S
## -5 -4 -3 -2 -1  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
##  T  U  V  W  X  Y  Z  [ \\  ]  ^  _  `  a  b  c  d  e  f  g  h  i
## 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41

fastq2.1 <- readFastq('../../data/fastq_files/fastq2_1.txt')
encoding(quality(fastq2.1))

##  !  "  #  $  %  &  '  (  )  *  +  ,  -  .  /  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
##  :  ;  <  =  >  ?  @  A  B  C  D  E  F  G  H  I  J
## 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41
```

Exercise (somewhat difficult): Generate a plot showing the average Phred score as a function of the position in the read.

3 Merging overlapping with paired reads

4 Reading BAM files

BAM files are compressed files that contain the information from the FASTQ files, plus additional information about the location where the reads map. A key feature of the BAM files is that they can be indexed, i.e. an associated file contains information about where each of the reads are located in the file. It allows very quick retrieval of reads that map to a given genomic location, which is the typical way one wants to use BAM files (for example, to extract all the reads that map to a gene of interest in order to find rare variants).

4.1 Using shell scripts

`samtools` is the key piece of software that is used to read, write and index BAM files. The manual page is the first place to go to find information about how to use `samtools`.

4.2 Using R and `Rsamtools`

The `Rsamtools` package in `R` is very effective to parse BAM files, and extremely memory efficient, making full used of BAM indexes. Look at the example below for example. Inspect the output object called `bam.reads`. Can you understand its structure? See what it contains and how the data are organised?

```
library(Rsamtools)
library(GenomicRanges)

which <- GRanges(seqnames=Rle('21'),
                 IRanges(start = 43000000, end = 45000000))

what <- c("rname", "strand", "pos", "qwidth", "seq")
param <- ScanBamParam(which=which, what=what)

bam.reads <- scanBam(file = '../data/BAM_files/HG00251.mapped.ILLUMINA.bwa.GBR.exome.20121211.bam',

## Error:  failed to open BamFile:  file(s) do not exist:
##  '../data/BAM_files/HG00251.mapped.ILLUMINA.bwa.GBR.exome.20121211.bam'

names(bam.reads[[1]])

## Error:  object 'bam.reads' not found
```

5 Session info

```
sessionInfo()

## R version 3.1.0 (2014-04-10)
## Platform: x86_64-unknown-linux-gnu (64-bit)
##
## locale:
##  [1] LC_CTYPE=en_US.iso885915       LC_NUMERIC=C
##  [3] LC_TIME=en_US.iso885915        LC_COLLATE=en_US.iso885915
##  [5] LC_MONETARY=en_US.iso885915    LC_MESSAGES=en_US.iso885915
##  [7] LC_PAPER=en_US.iso885915       LC_NAME=C
##  [9] LC_ADDRESS=C                   LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.iso885915 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  methods   stats     graphics  grDevices utils     datasets
## [8] base
##
## other attached packages:
## [1] ShortRead_1.20.0     Rsamtools_1.14.3     lattice_0.20-29
## [4] Biostrings_2.30.1    GenomicRanges_1.14.4 XVector_0.2.0
## [7] IRanges_1.20.7       BiocGenerics_0.8.0   knitr_1.6
##
## loaded via a namespace (and not attached):
##  [1] Biobase_2.22.0      bitops_1.0-6        evaluate_0.5.5
##  [4] formatR_0.10        grid_3.1.0          highr_0.3
##  [7] hwriter_1.3         latticeExtra_0.6-26 RColorBrewer_1.0-5
## [10] stats4_3.1.0        stringr_0.6.2       tools_3.1.0
## [13] zlibbioc_1.8.0
```