BCGES short courses, session 1: Introduction, file formats, shell scripting, Galaxy

Vincent Plagnol

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 (1h)

1.1 Basic shell scripting to read NGS files (20 minutes)

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^^^^^BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
## @A81CH8ABXX:4:1101:1583:1836#NGACCAAT/1
## NTCCCCAAAGCACAGGGCTCAGCTCCAGAGGGAGACGGGCTGGGCTGTCAGCGGGCCCAGGGGCACGCCACTGTTCAGAACAACTGGTTG
## +
## BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
## @A81CH8ABXX:4:1101:1729:1852#TGACCAAT/1
## NCTATGGACTGTGGTAAAGCTAGGATTAGTAACCAGACATTACTTACCTTGGCTCCGATCTGGTTGCCACACTGGCCAATCTGAATATGG
```

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

## +
## ggggggggggggggggggggggggggggggggggggggggggggggggggggggggggegfgggggaeeede[^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
## +
## gggggfgggggggggggdfgdgfggggggggffggggdgggggggfgfggfbgggggggggggfgegggcgg^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 different 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 (15 minutes)

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')
```

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. A lot of 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" "A81CH8ABXX:4:1101:1583:1836#NGACCAAT/1"
## [3] "A81CH8ABXX:4:1101:1729:1852#TGACCAAT/1" "A81CH8ABXX:4:1101:1642:1867#TGACCAAT/1"
## [5] "A81CH8ABXX:4:1101:1715:1891#TGACCAAT/1" "A81CH8ABXX:4:1101:1624:1941#TGACCAAT/1"
```

It is also possible to use this package to look at quality scores. 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^^^^^BBBBBBB...BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
##    [2]    90 BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB...BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
##    [3]    90 BWSSQVUVUTTQVSVUUUWW___BBBBBBBBBBBBBBBBBBB...BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
##    [4]    90 BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB...BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
##    [5]    90 gefgggggdgegfgggdfegee`eedddbdfZeffde^dBBB...BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
##    ...   ... ...
## [2496]    90 gfggggggegggggggfggefefggfcgafcdebgfeggad...eee_ddeabdb_cedeed0Z^``IZXYW^\YW]`ccX_adbdb
## [2497]    90 gggggggggggggfgggggggdgfgcgggfgfgggggfgg...bggfgefggggegecgggegdfeeeg^dbdee`bNddeee^a
## [2498]    90 gggggggggggggggggggggggggggggggggggggggggg...egfgggggaeeede[^d[_Y``b]a[bZXZ[Y`U^BBBBBBB
## [2499]    90 eedeee]decdbdbacca`bdbbdac_adWdb__dcee\cee...ee_ede^ddbdaUWbbd__aa[dee`e\bb_cad`dZcZc\b
## [2500]    90 gggggfggggggggggdfgdgfgggggggffggggdggggg...fbgggggggggggfgegggcgg^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  T  U  V  W  X  Y  Z  [
## -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 20 21 22 23 24 25 26 27
## \\  ]  ^  _  `  a  b  c  d  e  f  g  h  i
## 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  :  ;  <  =  >  ?  @  A
##  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 30 31 32
##  B  C  D  E  F  G  H  I  J
## 33 34 35 36 37 38 39 40 41
```

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

1.3 Using the Galaxy server (30 minutes)

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 propose that you use the main Galaxy instance which I know works for the purpose of this exercise. However, you can try different options but be ready for small and subtle differences and be flexible associated with different versions of the software.
Exercise: Create an account on one of these Galaxy servers and upload the pair of fastq *fastq2_1.txt* and *fastq2_2.txt*. Run the `fastqc` code on these files (note that a standalone version also exists and can be found here).
(Optional) Exercise: Run `fastqc` locally (after downloading it from the web) and compare the output with what you get from Galaxy (it should be identical!).

2 Reading and manipulating SAM, BAM and CRAM files using `samtools` (about 1.5h)

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). CRAM provides most of these features but in a more effective compression format. This is the future, if we can just manage to upgrade our tools.

`samtools` is the key piece of software that is used to read, write and index BAM files. `samtools` has been maintained essentially by Heng Li until the release of version 1.0 However, some tools used in this class still rely on the "old" samtools. For that reason you should find two versions of `samtools` installed.

- The old version which is in your path and simply called `samtools`. This one will disappear in the next iteration of the class, but it is still here for the time being.
- The updated version 1.2 of `samtools`, for which you can find the manual at this location and can be called using `samtools1.2`. This is the version with CRAM support.

So in most cases, rather than `samtools` you probably want to use `samtools1.2`, though many functions are equivalent.

2.1 Converting between formats (10 minutes)

One thing one does extensively with sequence data is shifting from one format to the next. Consider for example this BAM file, which has been provided as an example BAM for the class:

```
ls -ltrh ../data/BAM_files/HG00130.mapped.ILLUMINA.bwa.GBR.exome.20130415.bam

## -rw-r--r--  1 vplagnol  staff   260K 30 Aug 21:54 ../data/BAM_files/HG00130.mapped.ILLUMINA.bwa.GBR.exo
```

We will have a go at the following **Exercise:** convert this file to CRAM and check that the compression has improved.

2.2 More `samtools` (40 minutes)

A very useful feature of `samtools` is that it can work over a ftp site, by downloading the index locally and only pulling the reads that are relevant. That allows to access rich datasets online without having to download very large files. The exercise below illustrates some of these features.
Exercise: The following should be useful as an introduction to the sort of things one may want to do with `samtools`. The manual page should have all the commands and ideas to go through these, so best to have a go and try.

1. Find the CRAM file for sample HG00118 on the ftp site. Look at this location.
2. Using `samtools view` (version 1.2 required for CRAM analysis) over the web, download the *BRCA1* reads for the sample `HG00118`. Make sure that your output is in BAM format.
   Note: This step may seem difficult at first so the resulting BAM file should be available by default as part of the github set of files.
3. Index this BAM file using `samtools index`.
4. Still using `samtools view`, subset the BAM file for the first coding exon (not the UTR) of *BRCA1* (use transcript ENST00000357654 from Ensembl) and output in SAM format.
5. Exercise: What do these lines do?

```
samtools1.2 view -f 0x0002 results/BRCA1_HG00118.bam | wc -l
samtools1.2 view -F 0x0002 results/BRCA1_HG00118.bam | wc -l
samtools1.2 view -F 0x0040 results/BRCA1_HG00118.bam | wc -l

##     9905
##      113
##     5018
```

2.3 Understanding the SAM/BAM headers (10 minutes)

Key information about a BAM file can be obtained from the headers. `Samtools` is once again the key tool to read these data. You can view these headers using:

```
samtools view -H results/BRCA1_HG00118.bam > results/headers.txt
```

```
less -S results/headers.txt
```

Exercise: Go through the output of `samtools1.2 view -H` and make sure you understand what all the fields mean. This will be discussed in class. In the case of the downloaded 1KG file, much processing has happened so not all fields will make sense, but the important thing is to understand the general philosophy of the file. The ID, LB, SM tags are particularly useful.

2.4 Using `R` and `Rsamtools` (20 minutes)

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? We will be using these tools later on in the CNV analysis section.

```
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",
                     param=param)
names(bam.reads[[1]])

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

head(bam.reads[[1]]$pos)

## [1] 43823883 43823889 43823889 43823894 43823897 43823904
```

3 Merging overlapping paired-end reads (20 minutes)

Another useful thing to be aware of is the possibility to merge read pairs that overlap in the middle. This happens when the combined length of both reads exceeds the length of the DNA fragment they originate from. A review of tools to perform this task is available here. I have tested a few of these tools and I have been happy with two of them:

- One is PEAR
- And the other is FLASH.

If time allows and this is something you are interested in, I propose to have a go at running `PEAR`. Start by downloading the pre-compiled binary to your own computer (this program is not preinstalled). You should be able to run something like the command below:

```
pear -e -v 10 -c 40 -f ../data/fastq_for_merging/reads1.fq \
                    -r ../data/fastq_for_merging/reads2.fq \
                    -o results/pear_merged
```

And now something very similar using `flash`:

```
flash -m 10 -o results/flash_merge ../data/fastq_for_merging/reads1.fq \
                                   ../data/fastq_for_merging/reads2.fq \
                                   1> results/flash.out 2> results/flash.error
```

Check what the output looks like, in particular the read length of the assembled reads. The example provided is meant to be an easy one, and all reads should assemble properly. Have a look at the options that `PEAR` and `FLASH` use. There are some discrepancies between the output of both tools. It may be interesting to understand them and assess which you think is doing the best job.

4 Trimming reads (20 minutes)

Trimming reads usually refers to the process of cutting the end of the reads, because these are more likely to be of lower quality. One good way to illustrate the importance of trimming is to follow-up on the example above of merging paired end reads but with a more difficult example.

Here is what it looks like with `flash`:

```
flash -m 10 -o results/tough_merge ../data/fastq_for_merging/tough_read1.fq \
                                   ../data/fastq_for_merging/tough_read2.fq \
                                   1> results/flash_tough.out 2> results/flash_tough.error
```

Exercise: Can you understand why the merging fails so miserably in this example? What is the source of the problem? (it is actually quite a NextSeq specific problem).

So the conclusion of the previous exercise is that the outcome looks terrible! Now we will want to fix this. My tool of choice is called `sickle` and it has not failed me so far. What we want to do is cut the right end of each read (and only the right end!) as long as the average quality is lower than 20.

```
sickle pe -t sanger -l 0 -q 20 -f ../data/fastq_for_merging/tough_read1.fq \
                                -r ../data/fastq_for_merging/tough_read2.fq \
                                -o results/tough_read1_trimmed.fq \
                                -p results/tough_read2_trimmed.fq \
                                -s no_use_because_l_is_zero.fq
```

Exercise: Can you now try to merge again using the trimmed files? How much better does it look? Do you understand where the difference comes from? What do you think the `awk` command line below is doing. `awk` is a very handy command line utility and I recommend getting used to it.

```
awk 'NR%4==2{sum+=length($0)}END{print sum/(NR/4)}' results/tough_read1_trimmed.fq
awk 'NR%4==2{sum+=length($0)}END{print sum/(NR/4)}' results/tough_read2_trimmed.fq
```

And now you can look at the output of the `flash` code, and see how much better things appear. Make sure you understand why that is the case.

Session info

```
sessionInfo()

## R version 3.1.3 (2015-03-09)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.10.5 (Yosemite)
##
## locale:
## [1] en_GB.UTF-8/en_GB.UTF-8/en_GB.UTF-8/C/en_GB.UTF-8/en_GB.UTF-8
##
## attached base packages:
## [1] stats4    parallel  methods   stats     graphics  grDevices utils     datasets  base
##
## other attached packages:
##  [1] ShortRead_1.24.0        GenomicAlignments_1.2.2 Rsamtools_1.18.3        GenomicRanges_1.18.4
##  [5] GenomeInfoDb_1.2.5      Biostrings_2.34.1       XVector_0.6.0           IRanges_2.0.1
##  [9] S4Vectors_0.4.0         BiocParallel_1.0.3      BiocGenerics_0.12.1     knitr_1.9
##
## loaded via a namespace (and not attached):
##  [1] base64enc_0.1-3     BatchJobs_1.6       BBmisc_1.9          Biobase_2.26.0
##  [5] bitops_1.0-6        brew_1.0-6          checkmate_1.6.2     codetools_0.2-10
##  [9] DBI_0.3.1           digest_0.6.8        evaluate_0.5.5      fail_1.2
## [13] foreach_1.4.2       formatR_1.0         grid_3.1.3          highr_0.4
## [17] hwriter_1.3.2       iterators_1.0.7     lattice_0.20-30     latticeExtra_0.6-26
## [21] RColorBrewer_1.1-2  RSQLite_1.0.0       sendmailR_1.2-1     stringr_0.6.2
## [25] tools_3.1.3         zlibbioc_1.12.0
```