A ChIP-Seq Data Analyis

19 November, 2009

Contents

1 Introduction

Our goal in this section of the course is to describe the use of Bioconductor software to perform some basic tasks in the analysis of ChIP-Seq data. We will use the chipseq, IRanges and ShortRead packages, and also use the lattice package for visualization.

```
> library(ChIPseqTutorial)
```

1.1 Example data

The `data` folder contains two data files, each containing information for three chromosomes from one Solexa lane, one from a mouse ChIP-seq experiment done with CTCF antibodies, and one with GFP antibodies (as a background control). The raw reads were aligned to the reference genome (mouse in this case) using an external program (MAQ), and the results read in using the `readAligned` function in the ShortRead package. All duplicate reads were removed and a quality score cutoff of 5 was used.

```
> data(ctcf, package="ChIPseqTutorial")
> data(gfp, package="ChIPseqTutorial")
```

`ctcf` and `gfp` are objects of class *AlignedRead*.

```
> ctcf
```

```
class: AlignedRead
length: 484957 reads; width: 24 cycles
chromosome: chr10 chr10 ... chr12 chr12
position: 3011944 3012936 ... 121253739 121255103
strand: - + ... + +
alignQuality: IntegerQuality
alignData varLabels: nMismatchBestHit mismatchQuality nExactMatch24 nOneMismatch24
```

```
> gfp
```

```
class: AlignedRead
length: 316176 reads; width: 24 cycles
chromosome: chr10 chr10 ... chr12 chr12
position: 3002512 3008979 ... 121255999 121256287
strand: + - ... + +
alignQuality: IntegerQuality
alignData varLabels: nMismatchBestHit mismatchQuality nExactMatch24 nOneMismatch24
```

Further information on each alignment can be obtained using various accessor functions whose names are hinted at in the summarized display. For example,

```
> table(strand(ctcf))
```

```
     -      +      *
240633 244324      0
```

```
> table(chromosome(gfp))
```

```
 chr10  chr11  chr12
104970 120707  90499
```

1.2 The mouse genome

The data we have refer to alignments to a genome, and only makes sense in that context. Bioconductor has genome packages containing the full sequences of several genomes. The one relevant for us is

```
> library(BSgenome.Mmusculus.UCSC.mm9)
> mouse.chromlens <- seqlengths(Mmusculus)
> head(mouse.chromlens)
```

```
     chr1      chr2      chr3      chr4      chr5      chr6
197195432 181748087 159599783 155630120 152537259 149517037
```

We will only make use of the chromosome lengths, but the actual sequence will be needed for motif finding, etc.

2 Coverage, islands, and depth

Extending reads Solexa gives us the first few (24 in this example) bases of each fragment it sequences, but the actual fragment is longer. By design, the sites of interest (transcription factor binding sites) should be somewhere in the fragment, but not necessarily in its initial part. Although the actual lengths of fragments vary, extending the alignment of the short read by a fixed amount in the appropriate direction, depending on whether the alignment was to the positive or negative strand, makes it more likely that we cover the actual site of interest. We will extend the aligned regions to a length of 150 bases.

The extended regions can be summarized by their *coverage*, that is, how many times each base in the genome was covered by one of them.

```
> cov.ctcf <- coverage(ctcf, width = mouse.chromlens, extend = 126L)
> cov.ctcf

SimpleRleList of length 3
$chr10
'integer' Rle of length 129993255 with 310773 runs
  Lengths:  3011817 150 882 86 5 3 3 2 6 8 ...
  Values :  0 1 0 1 2 3 4 5 6 7 ...

$chr11
'integer' Rle of length 121843856 with 389113 runs
  Lengths:  3001523 150 100 150 182 150 296 150 372 150 ...
  Values :  0 1 0 1 0 1 0 1 0 1 ...

$chr12
'integer' Rle of length 121257530 with 258228 runs
  Lengths:  3015326 150 2738 150 25008 150 2242 150 11892 150 ...
  Values :  0 1 0 1 0 1 0 1 0 1 ...

> cov.ctcf$chr10

'integer' Rle of length 129993255 with 310773 runs
  Lengths:  3011817 150 882 86 5 3 3 2 6 8 ...
  Values :  0 1 0 1 2 3 4 5 6 7 ...
```

For efficiency, the result is stored in a run-length encoded (*Rle*) form.

The regions of interest are contiguous segments of non-zero coverage, also known as *islands*. Islands can be identified by *slicing* the coverage at a depth of 1:

```
> islands <- slice(cov.ctcf$chr10, lower = 1)
> islands

Views on a 129993255-length Rle subject
```

```
views:
            start       end width
    [1]   3011818   3011967   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [2]   3012850   3013220   371 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [3]   3018464   3018613   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [4]   3020766   3020915   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [5]   3023019   3023168   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [6]   3023240   3023494   255 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [7]   3032586   3032735   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [8]   3037521   3037670   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    [9]   3038377   3038526   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
    ...       ...       ...   ... ...
[99715] 129973225 129973457   233 [1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ...]
[99716] 129974863 129975012   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[99717] 129975575 129975724   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[99718] 129978669 129978818   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[99719] 129979259 129979521   263 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[99720] 129980303 129980452   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[99721] 129981957 129982106   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[99722] 129982380 129982529   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
[99723] 129987020 129987169   150 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...]
```

For each island, we can compute its area, i. e. the sum of the coverage values within the island, and the maximum coverage value (here, we use the function `head` to display results only for the first few islands).

```
> viewSums(head(islands))

[1]  150 2100  150  150  150  300

> viewMaxs(head(islands))

[1]  1 13  1  1  1  2

> nread.tab <- table(viewSums(islands) / 150L)
> depth.tab <- table(viewMaxs(islands))
> head(nread.tab, 10)

    1     2     3     4     5     6     7     8     9    10
80172 13548  2756   797   324   209   185   119   116    93

> head(depth.tab, 10)

    1     2     3     4     5     6     7     8     9    10
80230 14750  2124   472   240   184   153   121   115   107
```

Exercise 1
Repeat these steps for the `gfp` dataset.

2.1 Processing multiple lanes

Although data from one chromosome within one lane is often the natural unit to work with, we typically want to apply any procedure to the data from all chromosomes and from all lanes. We can recursively apply a summary function to all chromosomes using the `lapply` function. Here is a simple summary function that computes the frequency distribution of the number of reads per island.

```
> islandReadSummary <- function(cov)
+ {
+     s <- slice(cov, lower = 1)
+     tab <- table(viewSums(s) / 150)
+     ans <- data.frame(nread = as.numeric(names(tab)),
+                       count = as.numeric(tab))
+     ans
+ }
```

Applying it to our test-case, we get

```
> head(islandReadSummary(cov.ctcf$chr10))
```

```
  nread count
1     1 80172
2     2 13548
3     3  2756
4     4   797
5     5   324
6     6   209
```

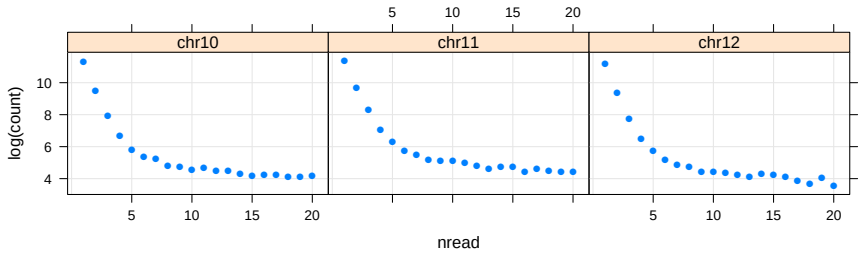
We can now use it to summarize the full dataset.

```
> nread.islands <- lapply(cov.ctcf, islandReadSummary)
> nread.islands <- do.call(make.groups, nread.islands)
> head(nread.islands)
```

```
        nread count which
chr10.1     1 80172 chr10
chr10.2     2 13548 chr10
chr10.3     3  2756 chr10
chr10.4     4   797 chr10
chr10.5     5   324 chr10
chr10.6     6   209 chr10
```

Note the use of the `make.groups` function from the lattice package, which combines several data frames into a single data frame that includes a further column `which` indicating which of the data frames each row came from.

```
> xyplot(log(count) ~ nread | which, data = nread.islands,
+        subset = (nread <= 20), pch = 16, type = c("p", "g"))
```

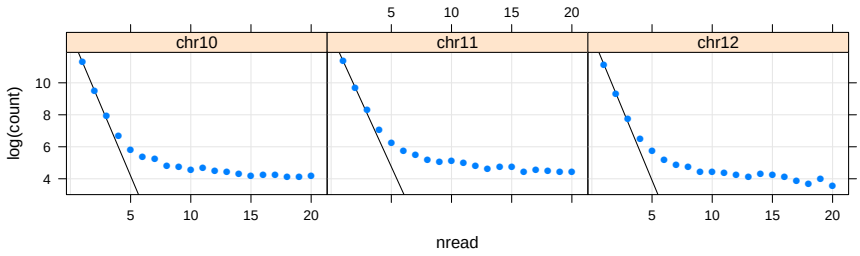


If reads were sampled randomly from the genome, then the null distribution of the number of reads per island would have a geometric distribution; that is,

$$P(X = k) = p^{k-1}(1 - p)$$

where p is the probability a randomly drawn read starts within a given interval of length 150. In other words, $\log P(X = k)$ is linear in k. Although our samples are not random, we can estimate p if we assume that the islands with just one or two reads are representative of the null distribution.

```
> xyplot(log(count) ~ nread | which, data = nread.islands,
+        subset = (nread <= 20),
+        pch = 16,
+        panel = function(x, y, ...) {
+            panel.grid(h = -1, v = -1)
+            panel.lmline(x[1:2], y[1:2], col = "black")
+            panel.xyplot(x, y, ...)
+        })
```

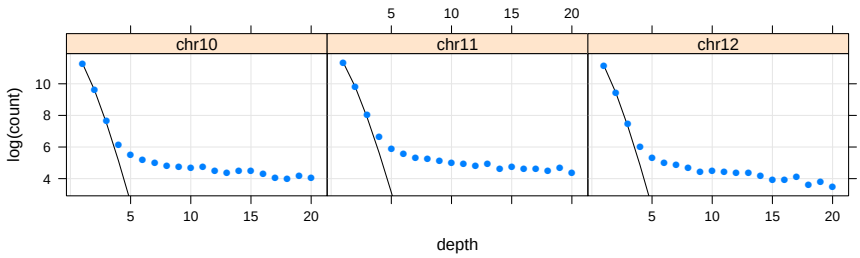


We can create a similar plot of the distribution of depths.

```
> islandDepthSummary <- function(cov)
+ {
+     s <- slice(cov, lower = 1)
+     tab <- table(viewMaxs(s))
+     ans <- data.frame(depth = as.numeric(names(tab)), count = as.numeric(tab))
+     ans
+ }
```

```
> depth.islands <- lapply(cov.ctcf, islandDepthSummary)
> depth.islands <- do.call(make.groups, depth.islands)
> xyplot(log(count) ~ depth | which, depth.islands,
+        subset = (depth <= 20), pch = 16,
+        panel = function(x, y, ...) {
+            panel.grid(h = -1, v = -1)
+            lambda <- 2 * exp(y[2]) / exp(y[1])
+            null.est <- function(xx) {
+                xx * log(lambda) - lambda - lgamma(xx + 1)
+            }
+            log.N.hat <- null.est(1) - y[1]
+            panel.lines(1:10, -log.N.hat + null.est(1:10), col = "black")
+            panel.xyplot(x, y, ...)
+        })
```

This assumes that the null distribution of depths has a Poisson distribution, which is not strictly true, but seems to give a reasonble fit.



Exercise 2
Produce similar plots for the `gfp` *dataset. What qualitatitve differences do you see? Based on your findings, what would be a reasonbale cutoff for deciding that the depth of an island is too high to be explained by chance, and hence is likely to contain a CTCF binding site?*

2.2 Peaks

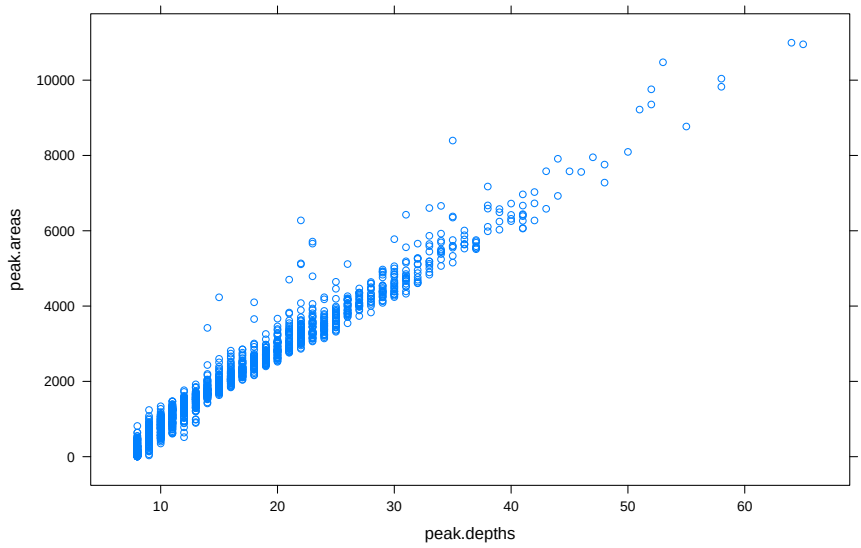
Going back to our example of chr10 of the first sample, let us define "peaks" to be contiguous regions of the genome where coverage is 8 or more. (More sophisticated, model-based or adaptive algorithms exist, and we refer to the literature in this active area of research).

```
> peaks <- slice(cov.ctcf$chr10, lower = 8)
> peaks

Views on a 129993255-length Rle subject

views:
           start       end width
```

```
   [1]   3012963   3013104   142 [ 8  8  8  8  9 10 11 11 11 11 11 11 11 ...]
   [2]   3234799   3234891    93 [ 8  8  8  8  8  8  8  8  8  8  8  8  8 ...]
   [3]   3270074   3270078     5 [8 8 8 8 8]
   [4]   3270083   3270260   178 [ 8  8  8  8  8  9  9  9  9  9  9 10 11 ...]
   [5]   3277683   3277816   134 [ 8  8  8  8  8  8  8  8  8  8  8  8  9 ...]
   [6]   3460866   3460928    63 [ 8  8  8  8  8  8  8  8  8  8  8  8  8 ...]
   [7]   3617844   3617947   104 [ 8  8  8  8  8  8  9  9  9  9  9  9  9 ...]
   [8]   3651762   3651972   211 [ 8  8 10 10 10 12 12 12 12 12 12 12 12 ...]
   [9]   4310429   4310675   247 [ 8  9  9 10 10 10 11 11 11 11 11 11 12 ...]
   ...       ...       ...   ... ...
[1746] 128986519 128986623   105 [8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 9 ...]
[1747] 128986644 128986647     4 [8 8 8 8]
[1748] 128986652 128986652     1 [8]
[1749] 129058941 129058952    12 [8 8 8 8 8 8 8 8 8 8 8 8]
[1750] 129530044 129530190   147 [ 8  8  8  8  8  8  8  8  9  9  9  9  9 ...]
[1751] 129533331 129533381    51 [8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 ...]
[1752] 129665388 129665570   183 [ 8  8  8  8  8  8  8  9 10 10 11 11 11 ...]
[1753] 129666765 129666905   141 [8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 9 ...]
[1754] 129750671 129750808   138 [ 8  8  8  8  8  8  9  9  9  9  9  9  9 ...]
```

Interesting properties of peaks are their maximum depth and area under the peak (a relative measure of how localized the peak is).

```
> peak.depths <- viewMaxs(peaks)
> peak.areas <- viewSums(peaks)
> xyplot(peak.areas ~  peak.depths)
```



Exercise 3
Produce a similar plot for the `gfp` *dataset. What differences do you see, particularly in terms of the number of peaks and the distribution of depths?*

We can order the peaks by depth

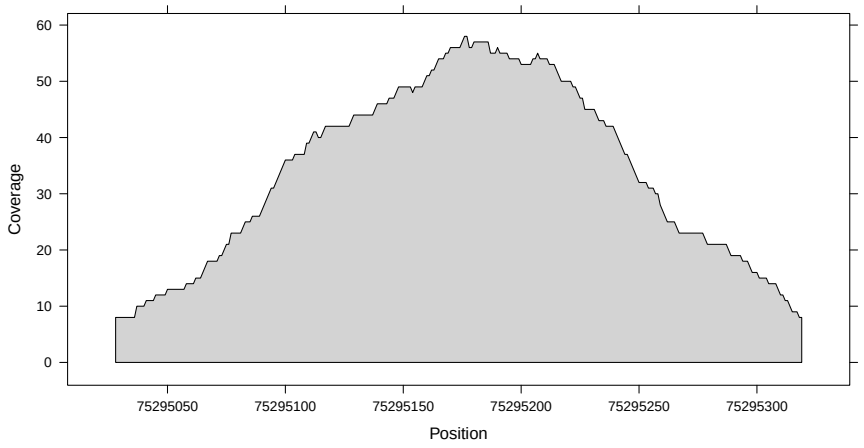
```
> wpeaks <- tail(order(peak.depths), 4)
> peaks[wpeaks]
```

```
Views on a 129993255-length Rle subject

views:
        start       end width
[1]  75295028  75295319   292 [ 8  8  8  8  8  8  8  8 10 10 10 10 11 ...]
[2] 126356178 126356472   295 [ 8  8  8  8  8  8  8 10 10 10 10 11 11 11 ...]
[3]  77875714  77876017   304 [ 8  8  9  9  9  9 10 10 11 10 10 10 10 10 ...]
[4]  80750534  80750831   298 [ 8  8  8  8  8  8  9 10 11 12 13 13 13 13 ...]
```

and plot individual peaks using this function:

```
> coverageplot <- function (peaks, xlab = "Position", ylab = "Coverage", ...)
+ {
+     pos1 <- seq(start(peaks[1]), end(peaks[1]))
+     cov1 <- as.integer(peaks[[1]])
+     pos1 <- c(head(pos1, 1), pos1, tail(pos1, 1))
+     cov1 <- c(0, cov1, 0)
+     xyplot(cov1 ~ pos1, ..., panel = panel.polygon,
+            col = "lightgrey", xlab = xlab, ylab = ylab)
+
+ }
```

```
> coverageplot(peaks[wpeaks[1]])
```



Exercise 4
How does the amount by which each read is extended affect the analysis? In calls to `coverage`*, we have used* `extend=126L` *to get a total length of 150 for each read. Try lengths of 100 and 200 and see how the results change.*

3 Differential expression

How can we determined which of the peaks present in the CTCF data are not present in GFP? Here is a list of all peaks, across all chromosomes

```
> allPeaks <- slice(cov.ctcf, lower = 8)
```

reads) under the peak

```
> ctcf.counts <- aggregate(cov.ctcf, allPeaks, sum)
```

The return value is a list-of-integers, each element of the list corresponding to a chromosome, and each member of the integer vector the number of reads under the corresponding peak; in your R session, display `ctcf.counts` to understand its structure. We can do the same for GFP

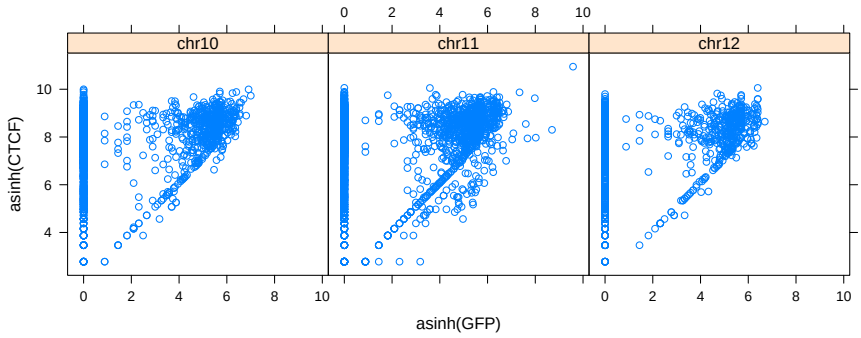
```
> cov.gfp <- coverage(gfp, width = mouse.chromlens, extend = 126L)
> gfp.counts <- aggregate(cov.gfp, allPeaks, sum)
```

To coordinate the data, lets put this into a RangedData object, converting the list-of-integers into a simple integer vector

```
> counts <- RangedData(allPeaks,
+                      CTCF=as.integer(ctcf.counts),
+                      GFP=as.integer(gfp.counts))
```

Finally we can display the counts. We do this by mapping the `RangedDatta` object into a standard `data.frame` to make it easier to use in standard R functions, transform the count data using the inverse hyperbolic sin (`asinh`) as a way of accomodating the non-normal distribution of the data, and use `xyplot` from lattice to plot the relationship between counts.

```
> xyplot(asinh(CTCF)~asinh(GFP)|space, as.data.frame(counts),
+        aspect="iso")
```



Each point represents a peak. Points on the diagonal correspond to peaks of comparable height in CTCF and GFP; these are not biologically interesting. Points parallel ot the y-axis at `asinh(GFP)` equal to 0 reprsent peaks present exclusively in the CTCF lane, whereas points above the diagonal represent points over-represented in CTCF; these are the biologically interesting peaks. We can retrieve these peaks with straight-forward operations, such as subsetting and ordering the data frame based on (a) absence of GFP and (b) number of reads in the CTCF lane. For instance, on chromosome 10 we have

```
> rd0 <- counts[counts[["GFP"]] == 0,]["chr10"]
> head(rd0[order(rd0[["CTCF"]], decreasing=TRUE),])
```

```
RangedData with 6 rows and 2 value columns across 1 space
        space                 ranges |      CTCF       GFP
  <character>              <IRanges> | <integer> <integer>
1       chr10 [ 77875714,  77876017] |     10994         0
2       chr10 [ 79627784,  79628107] |     10475         0
3       chr10 [ 79388076,  79388383] |      9757         0
4       chr10 [ 76298494,  76298756] |      8769         0
5       chr10 [ 77090319,  77090577] |      8095         0
6       chr10 [121488939, 121489225] |      7912         0
```

Exercise 5

There are a number of avenues for exploration at this point. The peaks can be exported to a genome browser (using rtracklayer's `export` function). Identified peaks can be annotated using Bioconductor annotation packages or ChIPseqAnno. Count data from multiple samples could be compared using the statistical analysis tools available in edgeR or baySeq. And importantly, alternative paths through the analysis can be readily explored. Pursue some of these suggestions!

4 Version information

- R version 2.10.0 Patched (2009-11-16 r50456), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=en_US.UTF-8`, `LC_MONETARY=C`, `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, graphics, grDevices, methods, stats, utils
- Other packages: Biostrings 2.14.5, BSgenome 1.14.1, BSgenome.Mmusculus.UCSC.mm9 1.3.16, chipseq 0.2.0, ChIPseqTutorial 0.0.1, IRanges 1.4.6, lattice 0.17-26, ShortRead 1.4.0
- Loaded via a namespace (and not attached): Biobase 2.6.0, grid 2.10.0, hwriter 1.1, tools 2.10.0