The rtracklayer package

Manipulating and visualizing genomic annotations

Michael Lawrence

May 30, 2009

Outline

Tracks and experimental data analysis

- Many data types have natural mapping to genome:
  - SNPs
  - Chip-seq peaks
  - Methylation
- Annotation databases contain wealth of knowledge:
  - Genes and exons (biomaRt)
  - Conservation scores
  - Transcription factor binding sites, TransFac

Tracks and experimental data analysis

- Many data types have natural mapping to genome:
  - SNPs
  - Chip-seq peaks
  - Methylation
- Annotation databases contain wealth of knowledge:
  - Genes and exons (biomaRt)
  - Conservation scores
  - Transcription factor binding sites, TransFac

Goal

Integrate the analysis of experimental data with existing annotations.

The rtracklayer package

The *rtracklayer* package is an interface (or *layer*) between **R**, genome browsers and genomic annotations.

Feature overview

- Annotation track representation and import/export (files and online databases)
- The control and querying of external genome browser sessions and views.
- Currently supports UCSC browser and database.

Outline

Storing data on intervals

The RangedData object

- *RangedData* objects, defined by the *IRanges* package, hold data on (genomic) intervals.
- Two components
  1. The interval starts and widths, segregated by chromosome
  2. The variables describing the intervals

Constructing a *RangedData*

- *rtracklayer* provides `GenomicData` for conveniently constructing *RangedData* tracks
- Here we store the peaks, and their area and depth.

Code

```
> peakTrack <- GenomicData(peaks, depth = peak.depths,
+                          area = peak.areas,
+                          chrom = "chr10",
+                          genome = "hg18")
> peakTrack

RangedData: 1754 ranges by 2 column(s) on 1 sequence(s)
columns(2): depth area
sequences(1): chr10
```

Accessing built-in attributes

Each built-in feature attribute has a corresponding accessor method: `start`, `end`, `strand`, `chrom`, `genome`

Example

```
> head(start(peakTrack))

[1]    1146 222982 258257 258266 265866
[6] 449049
```

Exercises

1. Get the width of each feature in the track
2. Get the genome for the track

Accessing built-in attributes

Each built-in feature attribute has a corresponding accessor method: `start`, `end`, `strand`, `chrom`, `genome`

Exercises

1. Get the width of each feature in the track

```
> head(width(peakTrack))
```

```
[1] 142  93   5 178 134  63
```

2. Get the genome for the track

Accessing built-in attributes

Each built-in feature attribute has a corresponding accessor method: `start`, `end`, `strand`, `chrom`, `genome`

Exercises

1. Get the width of each feature in the track

```
> head(width(peakTrack))
```

```
[1] 142  93   5 178 134  63
```

2. Get the genome for the track

```
> genome(peakTrack)
```

```
[1] "hg18"
```

Accessing data columns

Any data column (including strand) is accessible via `$` and `[[`.

Example

```
> head(peakTrack$area)
```

```
[1] 1664  856   40 2811 1374  573
```

Exercise

Construct a *data.frame* with peak starts, widths, areas and depths.

Accessing data columns

Any data column (including strand) is accessible via `$` and `[[`.

Example

```
> head(peakTrack$area)

[1] 1664  856   40 2811 1374  573
```

Exercise

Construct a *data.frame* with peak starts, widths, areas and depths.

```
> data.frame(start = start(peakTrack),
+            width = width(peakTrack),
+            area = peakTrack$area,
+            depth = peakTrack$depth)
> ## easier:
> as.data.frame(peakTrack)
```

Overview of *RangedData* subsetting

- Often need to subset track features and data columns
- Example: limit the amount transferred to a genome browser
- Matrix style: `track[i, j]`, where `i` is feature index and `j` is column index
- By chromosome: `track[i]`, where `i` indexes the chromosome

Subsetting examples and exercises

Examples

```
> ## get the first 10 targets
> first10 <- peakTrack[1:10,]
> ## get peaks of depth > 12
> highPeaks <- peakTrack[peakTrack$depth > 12,]
> ## get first (and only) chromosome
> chrPeaks <- peakTrack[1]
```

Exercise

Subset for area $>$ 1000 and discard depth column.

Subsetting examples and exercises

Examples

```
> ## get the first 10 targets
> first10 <- peakTrack[1:10,]
> ## get peaks of depth > 12
> highPeaks <- peakTrack[peakTrack$depth > 12,]
> ## get first (and only) chromosome
> chrPeaks <- peakTrack[1]
```

Exercise

Subset for area $>$ 1000 and discard depth column.

```
> peakTrack[peakTrack$area > 1000,"area"]
```

Overview of import/export

- Supported formats
  - BED Browser Extended Display, display-oriented, native format of UCSC
  - WIG Wiggle, sparse format for quantitative data
  - GFF General Feature Format (versions 1, 2, and 3), general storage, popular at EBI
- Functions: `import` and `export`
- Extensible via plugin system

Import/export examples and exercises

Examples

```
> export(peakTrack, "peaks.bed")
> restoredTrack <- import("peaks.bed")
> ## as character vector
> peaksChar <- export(peakTrack, format = "gff1")
```

Exercises

1. Output the track to a file in the "gff" format.
2. Read the track back into R.

Import/export examples and exercises

Examples

```
> export(peakTrack, "peaks.bed")
> restoredTrack <- import("peaks.bed")
> ## as character vector
> peaksChar <- export(peakTrack, format = "gff1")
```

Exercises

1. Output the track to a file in the "gff" format.

   ```
   > export(peakTrack, "peaks.gff")
   ```

2. Read the track back into R.

Import/export examples and exercises

Examples

```
> export(peakTrack, "peaks.bed")
> restoredTrack <- import("peaks.bed")
> ## as character vector
> peaksChar <- export(peakTrack, format = "gff1")
```

Exercises

1. Output the track to a file in the "gff" format.

   ```
   > export(peakTrack, "peaks.gff")
   ```

2. Read the track back into R.

   ```
   > peakGff <- import("peaks.gff",
   +                   genome = "hg18")
   ```

Outline

The genome browser interface

- *rtracklayer* interfaces with the UCSC genome browser
- Easily extended to support other browsers
- Workflow
  1. Start a browser session
  2. Load one or more tracks
  3. Open one or more browser views of specific regions
  4. Possibly download interesting annotations into R

Starting a browser session

Code

```
> session <- browserSession("UCSC")
```

The `session` object is a *BrowserSession* instance. With a session object, one may:

- Upload and download tracks to/from the genome browser
- Create browser views

The argument `"UCSC"` creates a session for the UCSC browser. To list all supported browsers:

Code

```
> genomeBrowsers()

[1] "UCSC"
```

Laying the target site track

Tracks may be loaded into a session with the `track<-`, `[[<-` and `$<-` functions.

Example

```
> track(session, "peaks") <- peakTrack
> ## equivalently
> session$peaks <- peakTrack
```

Exercise

Lay a track with the four heighest peaks

Laying the target site track

Tracks may be loaded into a session with the `track<-`, `[[<-` and `$<-` functions.

Example

```
> track(session, "peaks") <- peakTrack
> ## equivalently
> session$peaks <- peakTrack
```

Exercise

Lay a track with the four heighest peaks

```
> session$topPeaks <- peakTrack[wpeaks,]
```

Choosing a region to view

- The `range` function returns an object representing the genomic range of a track
- Assume we want to view the coverage of the tallest peak
  1. Get the range of the feature
  2. Zoom out by a factor of 10

Choosing a region to view

- The `range` function returns an object representing the genomic range of a track
- Assume we want to view the coverage of the tallest peak
  1. Get the range of the feature

Code

```
> high <- tail(wpeaks, 1)
> region <- range(peakTrack[high,])
```

  2. Zoom out by a factor of 10

Choosing a region to view

- The `range` function returns an object representing the genomic range of a track
- Assume we want to view the coverage of the tallest peak
  1. Get the range of the feature
  2. Zoom out by a factor of 10

Code

```
> region <- region * -10
```

Viewing the coverage

Load coverage track

```
> covTrack <- as(cov.ctcf$chr10, "RangedData")
> names(covTrack) <- "chr10"
> session$coverage <- covTrack
```

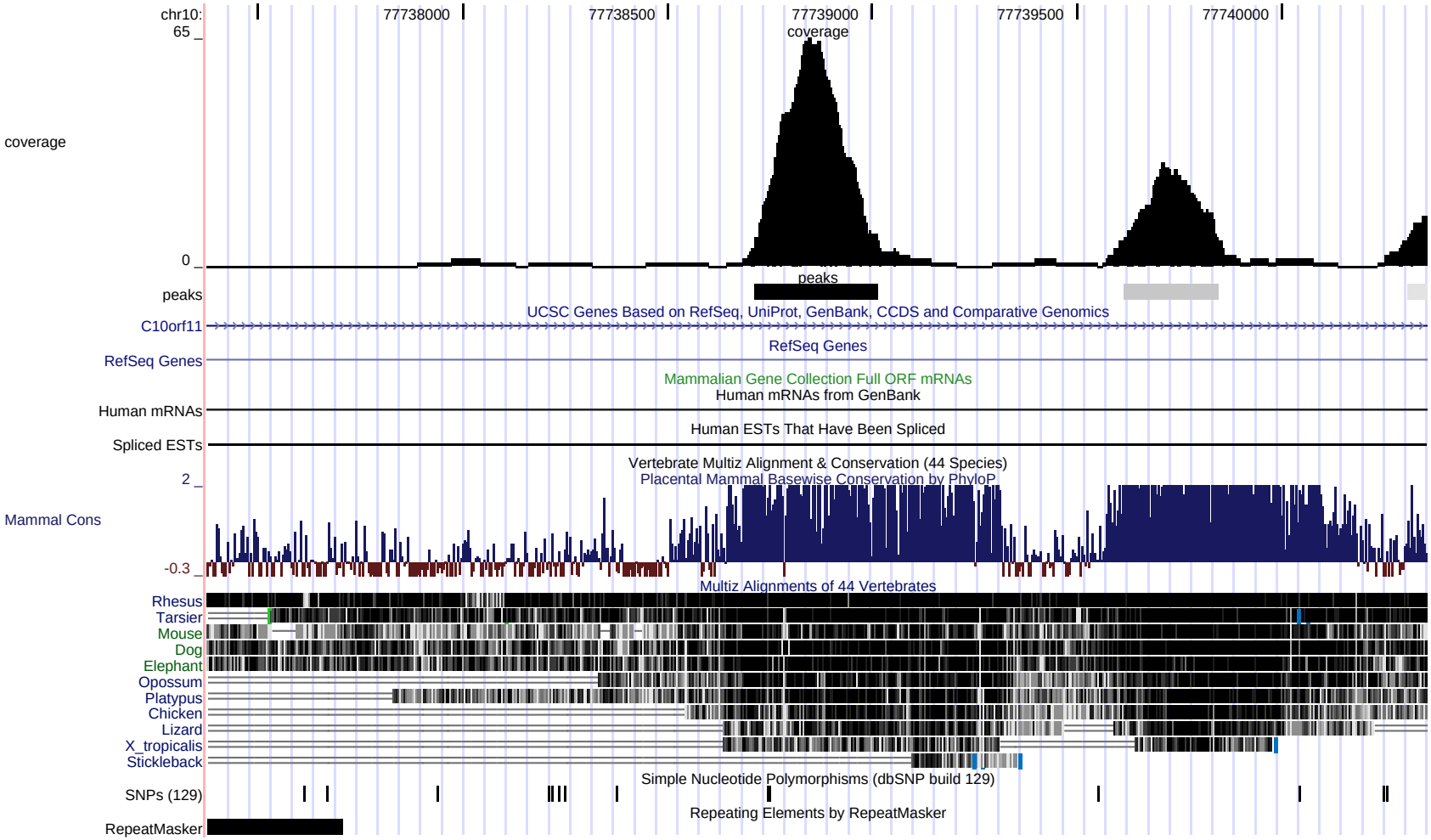
Display the view

```
> view <- browserView(session, region,
+                     full = "coverage")
```

The `view` object is a *BrowserView* instance. With a view object, one may:

- Change the currently visible region (pan/zoom)
- Change the visibility of tracks (show/hide)

Coverage view



View exercise

Exercise

Create a view with the same region as `view`, zoomed out 2X.

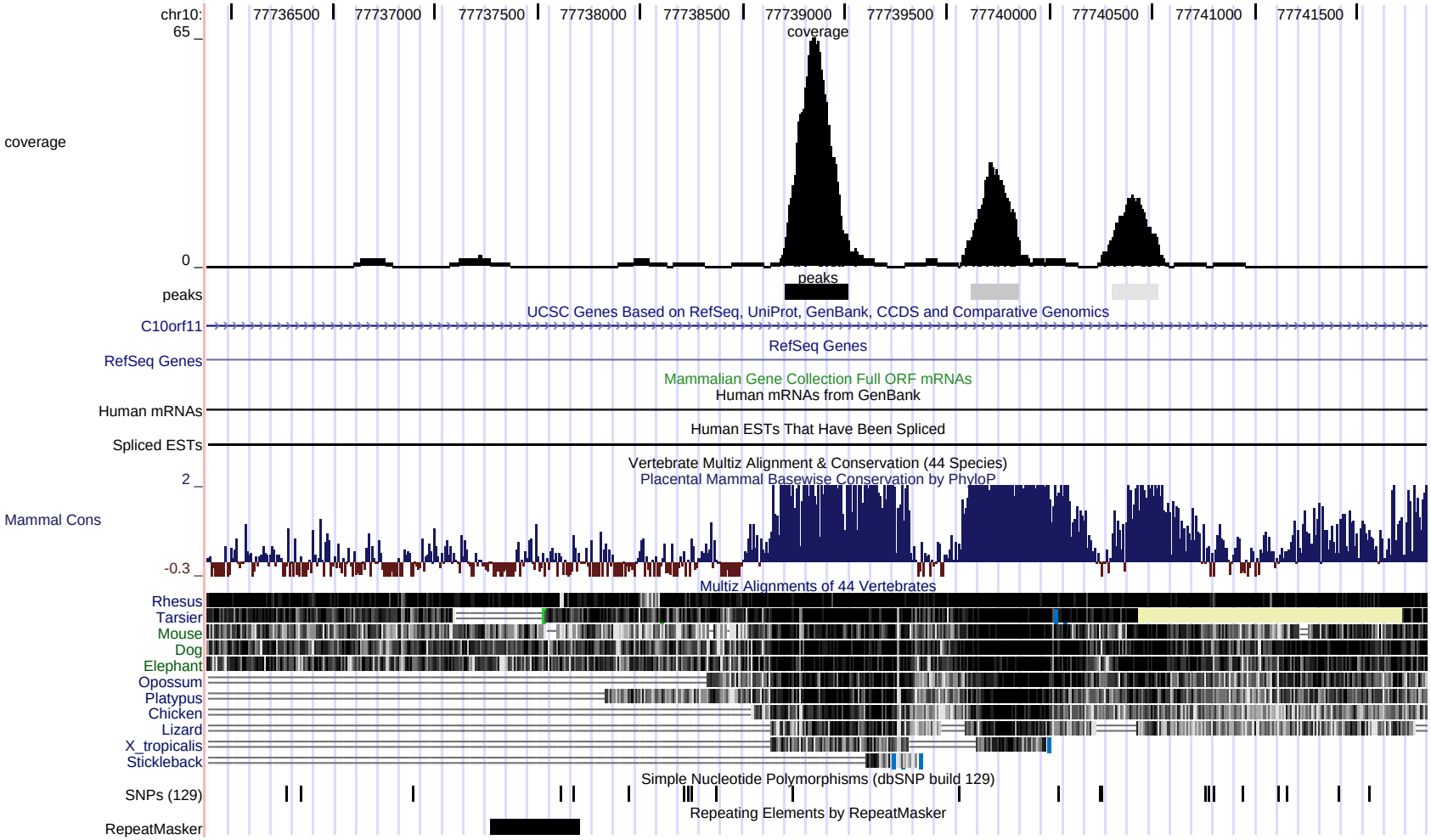
View exercise

Exercise

Create a view with the same region as `view`, zoomed out 2X.

```
> viewOut <- browserView(session, range(view) * -2)
```

Coverage view, zoomed out 2X



A shortcut

All of the above in a single step:

```
> browseGenome(peakTrack,
+              range = range(peakTrack[high,]) * -10)
```

A session is started, the track is loaded and a view is created around the first target site.

Changing view range

The `range<-` function sets a new visible range on a view.

Example

```
> ## zoom in 2X
> range(view) <- range(view) * 2
```

Exercise

Shift a view to the second highest peak

Changing view range

The `range<-` function sets a new visible range on a view.

Example

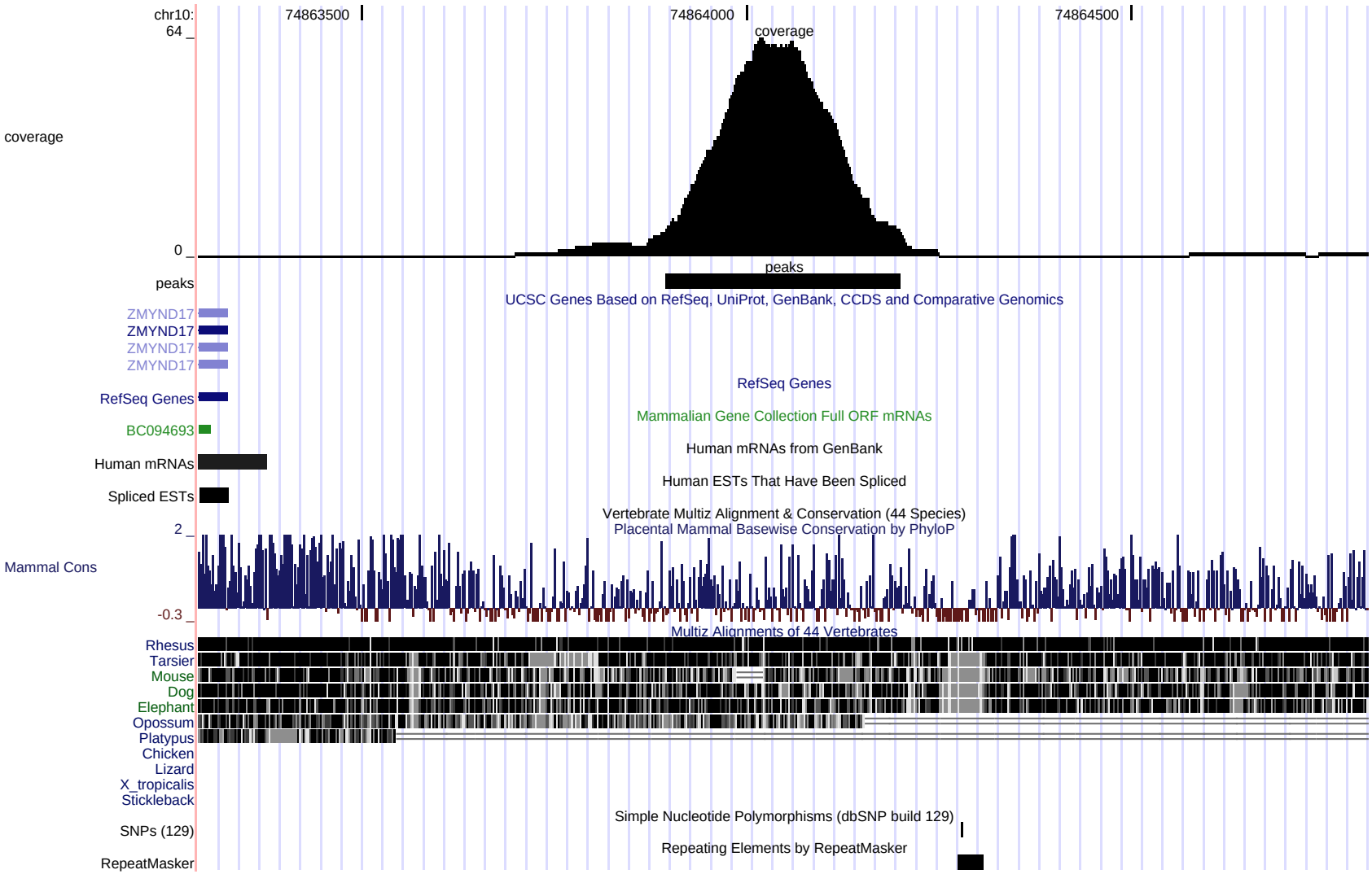
```
> ## zoom in 2X
> range(view) <- range(view) * 2
```

Exercise

Shift a view to the second highest peak

```
> second <- tail(wpeaks, 2)[1]
> range(viewOut) <- range(peakTrack[second,]) * -5
```

Second highest peak



Changing track visibility

Tracks may be shown or hidden with the `visible<-` function.

Example

```
> ## hide the Conservation track
> visible(view)["Conservation"] <- FALSE
```

Exercise

Make the "Ensembl Genes" track visible

Changing track visibility

Tracks may be shown or hidden with the `visible<-` function.

Example

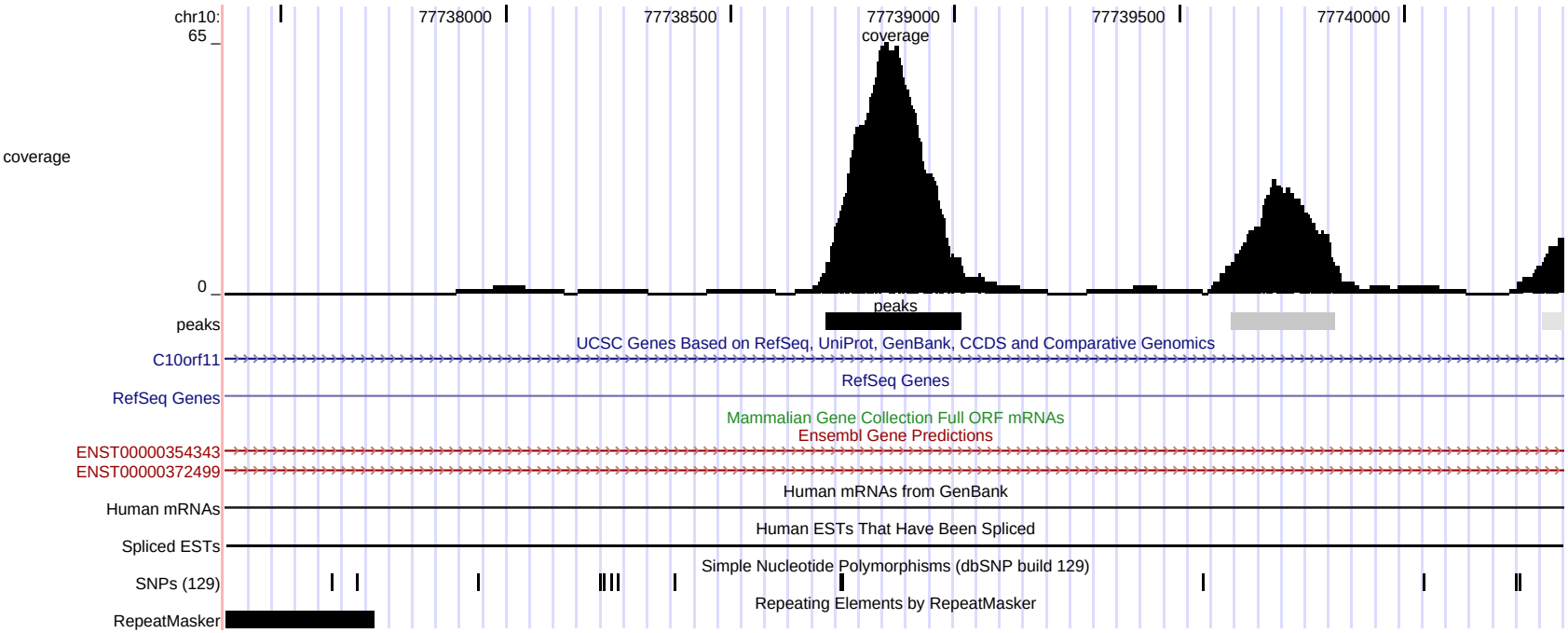
```
> ## hide the Conservation track
> visible(view)["Conservation"] <- FALSE
```

Exercise

Make the "Ensembl Genes" track visible

```
> visible(view)["Ensembl Genes"] <- TRUE
```

Customized view



Overview

- Many browsers are built upon large databases
- Often want to incorporate the data into an R analysis
- For UCSC, this interacts with the table browser

Retrieving browser tracks

1. List available tracks
2. Download named track (e.g. “Conservation”) in currently viewed region

Retrieving browser tracks

1. List available tracks

Code

```
> head(trackNames(session))

     coverage          peaks
"ct_coverage"      "ct_peaks"
Base Position Chromosome Band
      "ruler"      "cytoBand"
  STS Markers     FISH Clones
     "stsMap"    "fishClones"
```

2. Download named track (e.g. "Conservation") in currently viewed region

Retrieving browser tracks

1. List available tracks
2. Download named track (e.g. "Conservation") in currently viewed region

Code

```
> cons <- track(session, "Conservation")
> ## or specific region
> cons <- track(session, "Conservation",
+               range(view) * 2)
> ## shortcut
> session$Conservation

UCSC track 'Primate Cons'
UCSCData: 1490 ranges by 1 column(s) on 1 sequence(s)
columns(1): score
sequences(1): chr10
```

Outline

Beyond rtracklayer

- *rtracklayer* operates in the context of genome browsers
- Bioconductor has other sources of annotations:
  - The annotation packages
  - biomaRt

Session info

```
> sessionInfo()

R version 2.10.0 Under development (unstable) (--)
i686-pc-linux-gnu

locale:
[1] C

attached base packages:
[1] stats     graphics  grDevices
[4] utils     datasets  methods
[7] base

other attached packages:
[1] rtracklayer_1.5.2
[2] RCurl_0.94-1
[3] BSgenome.Mmusculus.UCSC.mm9_1.3.11
[4] ShortRead_1.3.6
[5] lattice_0.17-25
[6] BSgenome_1.13.4
[7] Biostrings_2.13.9
[8] IRanges_1.3.18

loaded via a namespace (and not attached):
[1] Biobase_2.3.11 XML_2.3-0
[3] grid_2.10.0    hwriter_1.1
```