ggbio: visualization toolkits for genomic data

Tengfei Yin

October 4, 2013

Contents

1 Getting started **4**

1.1 Citation 4

1.2 Introduction 5

1.3 Documentation 5

1.4 Support 5

1.5 Installation 5

1.6 Getting started 6

1.6.1 Genesis: everything started from *GRanges* 6

1.6.2 About *GRanges* 6

1.6.3 Visualize *GRanges* object 8

2 Visaulize gff-like files **18**

3 Visualize bam files **19**

4 How to make tracks **20**

4.1 Motivation 20

4.2 Minimal examples for `tracks` 22

5 Visualize single chromosome **26**

5.1 Introduction 26

5.2 Single chromosome visualization 26

5.2.1 Single chromosome use to be embedded in tracks. 26
5.2.2 Get ideogram or customize the colors 38

6 Circular view **40**

6.1 Introduction 40
6.2 Tutorial 40
6.2.1 Step 1: understand the layout circle 40
6.2.2 Step 2: get your data ready to plot 41
6.2.3 Step 3: low level API: `layout_circle` 43
6.2.4 Step 4: Complex arragnment of plots 49
6.3 Transform space 52

7 Manhattan plot **54**

7.1 Introduction 54
7.2 Understand the new coordinate 54
7.3 Step 2: Simulate a SNP data set 56
7.4 Step 3: Start to make Manhattan plot by using `autoplot` 57
7.5 Convenient `plotGrandLinear` function 58
7.6 Annotating manhattan plot easily 64
7.7 Unequal space 66

8 Karyogram overview **68**

8.1 Introduction 68
8.2 `autoplot` 68
8.3 `plotKaryogram` 73
8.4 `layout_karyogram` 73

9 Visualize genomic features **79**

9.1 Introduction 79
9.2 Usage 80

9.2.1 autoplot 80
9.2.2 geom_alignment 87

10 Visualize sequence **89**

11 Visualize matrix-related objects **90**

12 Visualize VCF files **91**

13 Visualize splicing events **92**

14 Miscellaneous **93**
14.1 Themes 93
14.1.1 Plot theme 93
14.1.2 Track theme 93
14.2 Scales 93

15 Session Information **94**

Chapter 1

Getting started

1.1 Citation

```
citation("ggbio")
```

```
##
## To cite package 'ggbio' in publications use:
##
##   Tengfei Yin, Dianne Cook and Michael Lawrence (2012): ggbio:
##   an R package for extending the grammar of graphics for
##   genomic data Genome Biology 13:R77
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {ggbio: an R package for extending the grammar of graphics for genomic data},
##     author = {Tengfei Yin and Dianne Cook and Michael Lawrence},
##     journal = {Genome Biology},
##     volume = {13},
##     number = {8},
##     pages = {R77},
##     year = {2012},
##     publisher = {BioMed Central Ltd},
##   }
```

1.2 Introduction

The *ggbio* package extends and specializes the grammar of graphics for biological data. The graphics are designed to answer common scientific questions, in particular those often asked of high throughput genomics data. Almost all core Bioconductor data structures are supported, where appropriate. The package supports detailed views of particular genomic regions, as well as genome-wide overviews. Supported overviews include ideograms and grand linear views. High-level plots include sequence fragment length, edge-linked interval to data view, mismatch pileup, and several splicing summaries.

1.3 Documentation

After Bioconductor 2.11, two kind of documentation are provided.

- Vignettes knited from sweave files.
- Another source is *ggbio* official websites, `http://tengfei.github.com/ggbio`, under *documentation* tab, Rd help manual is knited to html webpages under manual section(`http://tengfei.github.com/ggbio/docs/man`), so all the help manual with examples code hybrided with graphics is shown there.

1.4 Support

For issue/bug report and questions about usage, you could

- File a issue/bug report at `https://github.com/tengfei/ggbio/issues`,
- Ask question about *ggbio* on biocondcutor mailing list.

1.5 Installation

As described on-line (`http://tengfei.github.com/ggbio/download.html`).

Tips: **github** is only used for issue/bugs report and homepage build purpose, developemnt has been stopped and removed from there already. I only use bioconductor to maintain and develop my package.

After R 2.15, R release cycle falls into annual release instead of semi-annual release cycle, at the same time, Bioconductor project still follows semi-annual release cycle. So now you can install both released and developmental version for the same version of R.

In your R session, please run following code to install released version of ggbio, but if you are using developmental version of R, you will get developmental version of ggbio automatically. Because what you get depends on the bioconductor installer, which is implemented in package BiocInstaller and its version decides which version of Bioconductor you got.

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

To install developmental version, run

```
library("BiocInstaller")
useDevel(TRUE)
biocLite("ggbio")
```

For developers, you can find latest source code in bioc svn.

1.6 Getting started

1.6.1 Genesis: everything started from *GRanges*

In our model, *GRanges* is the core data structure that support most direct geom/stat/layout transformation and visualization support, every other data structure always converted to *GRanges* first inside, and arrange components properly to bring some nice default graphics.

1.6.2 About *GRanges*

GRanges object is a container holding genomic interval data associated with meta data information. The power about *ggbio* is about flexible mapping for all those information.

Here is an example of *GRanges* and how to construct it by using constructor `GRanges`. We construct a *GRanges* object with three chromosomes named *chr1, chr2, chr3* and with seqlengths 400, 500, 1000. Pay attention to the seqlengths, if you didn't assign any value, these fields will be `NA`. And these are important information if you want to generate overview in genome space context later.

```
library(GenomicRanges)
set.seed(1)
N <- 100
gr <- GRanges(seqnames = sample(c("chr1", "chr2", "chr3"),
```

```
                    size = N, replace = TRUE),
             IRanges(start = sample(1:300, size = N, replace = TRUE),
                     width = sample(70:75, size = N,replace = TRUE)),
             strand = sample(c("+", "-"), size = N, replace = TRUE),
             value = rnorm(N, 10, 3), score = rnorm(N, 100, 30),
             sample = sample(c("Normal", "Tumor"),
               size = N, replace = TRUE),
             pair = sample(letters, size = N,
               replace = TRUE))
seqlengths(gr) <- c(400, 1000, 500)
head(gr)
```

```
## GRanges with 6 ranges and 4 metadata columns:
##       seqnames     ranges strand |     value     score      sample
##          <Rle>  <IRanges>  <Rle> | <numeric> <numeric> <character>
##   [1]     chr1 [197, 267]      - |    11.228    126.81       Tumor
##   [2]     chr2 [106, 176]      + |    15.067     68.58      Normal
##   [3]     chr2 [ 82, 154]      + |    14.760    159.14       Tumor
##   [4]     chr3 [298, 368]      + |     9.007     88.49       Tumor
##   [5]     chr1 [191, 261]      + |     3.144    149.62      Normal
##   [6]     chr3 [ 64, 136]      - |    17.493    145.37       Tumor
##              pair
##       <character>
##   [1]           v
##   [2]           t
##   [3]           h
##   [4]           e
##   [5]           f
##   [6]           b
##   ---
##   seqlengths:
##    chr1 chr2 chr3
##     400 1000  500
```

The first three columns are required information about intervals, including seqnames(chromosome names), ranges(interval start and end), strand(direction:*, +, -).

> Tips: For more information, please visit vignettes for package *IRanges, GenomicRanges*. Those packages provide awesome computational methods working on interval data, and have lots of convenient accessors, so we won't spend time introducing those tips here.

1.6.3 Visualize *GRanges* object

`autoplot` is the generic function which support most core Bioconductor objects, try to make different types of graphics for specific object.

```
library(ggbio)
autoplot(gr)
```

To set arbitrary aesthetics, such as color, size, etc.

```
autoplot(gr, color = "gray40", fill = "skyblue")
```

To map variables to certain aesthetics, **DON'T** forget to use `aes()` to wrap around the mapping, that's different with *ggplot2*'s `qplot` strategy. For example, if you want to map 'strand' variable to color, you have to put the mapping inside `aes()` and remember don't use quotes around the variable name.

```
autoplot(gr, aes(color = strand, fill = strand))
```

You could also pass 'facets' argument in `autoplot`, to split the data based on some column factors, use the form 'a ~ b', 'a' indicates the row and 'b' indicates the column.

> Tips: For implementation reason, if you pass facets inside `autoplot` that will usually work as expected, if you plus `facet_grid` and `facet_wrap` in the end of `autoplot`, for specific stat that won't work as expected. Because data are calculated split based facet formula and for now won't work in *ggplot2* evaluation fashion.

```
autoplot(gr, aes(color = strand, fill = strand), facets = strand ~ seqnames)
```

stat represents the statistical transformation from original data, allow you to plot or map new computed variable in the graphics. Default stat is 'stepping' which, as you have seen, print all the interval stacked upon each other without overlapping, we could try use other different *stat*, to specify it in the `autoplot` function. For example `stat_coverage`.

```
autoplot(gr, aes(color = strand, fill = strand), facets = strand ~ seqnames,
         stat = "coverage")
```

Some stats are very useful for summary statistics, for example, `stat_aggregate`.

```
autoplot(gr, stat = "aggregate", aes(y = score))
autoplot(gr, stat = "aggregate", aes(y = score), geom = "boxplot", window = 50)
```

coordinate is not a new idea, we all familiar with x-y Cartesian coordinates. We introduced new 'genome' coordinate in *ggbio*, that put all chromosomes together in a grand linear manner and relabel them only by chromosome names.

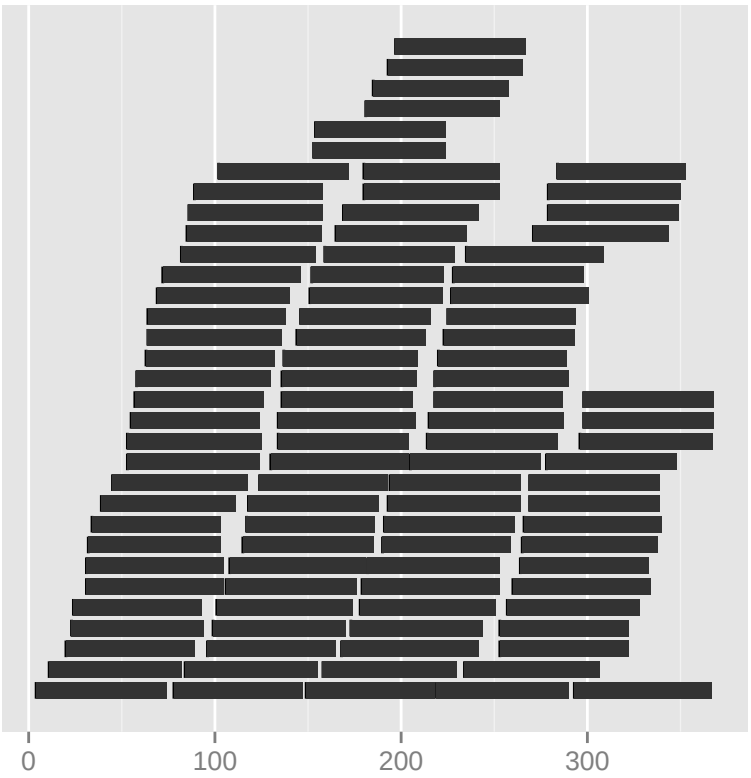
layout is a fairly new idea in *ggbio* which not exists in *ggplot2*, it's about how we layout the genome, in a circular fashion or in a karyogram fashion.

```
autoplot(gr, layout = "circle", aes(fill = seqnames))
```

```
autoplot(gr, coord = "genome")
```

The power about `autoplot` is not only for *GRanges*, but also for some other core Bioconductor data structures for example, *IRanges* object visualization strategy is almost identical to *GRanges*, except that those plots are not faceted by seqnames by default.

```
## For IRanges
autoplot(ranges(gr))


## Object of class "ggbio"
```




```
## For seqinfo
```

```
autoplot(seqinfo(gr))
autoplot(gr, layout = "karyogram", aes(fill = score))
```

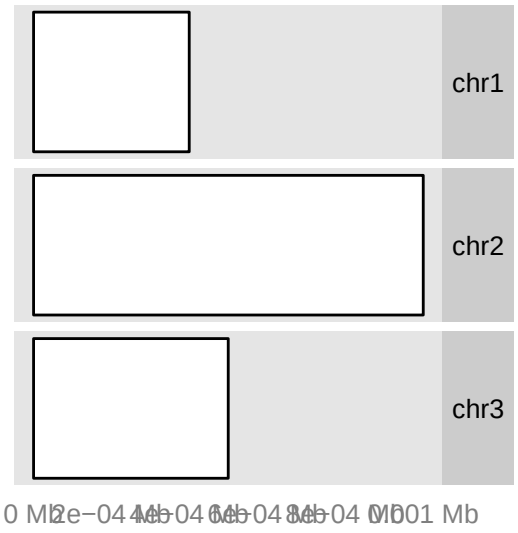

chr1
chr2
chr3
0 Mb
2e−04 Mb
4e−04 Mb
6e−04 Mb
8e−04 Mb
0.001 Mb

Table 1.1 shows objects we currently supported and following chapters will cover most of those topics.

Object	meanings	chapter
GRanges	Genomic interva	1.6.1
IRanges	numeric interval	1.6.1
GRangesList	List of genomic interval	1.6.1
Seqinfo	Information about genomic sequence	8
GappedAlignments	NGS data	3
BamFiles	Bam files container	3
character	Bam files path	3 2
BSgenome	Nucleotide sequence	10
matrix	matrix	11
Rle	Numeric vector	11
RleList	List of numeric vector	11
Views	Containter for a set of Views	11
ExpressionSet	Container for microarray data	11
SummarizedExperiment	eSet-like container	11
VCF	Containter for VCF format data	12

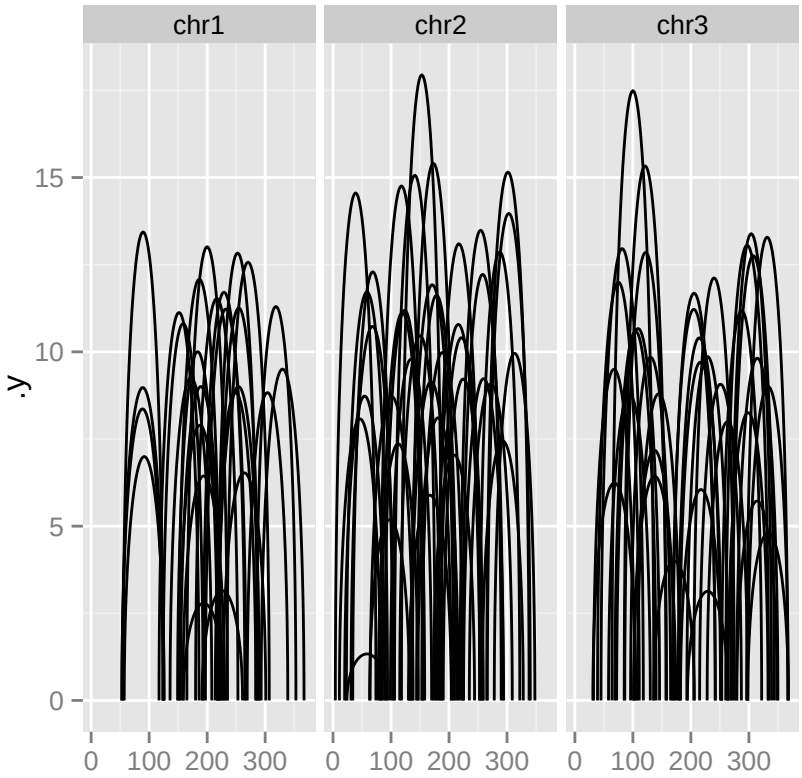
Table 1.1: Objects that `autoplot` supported.

Thouth `autoplot` is a very conventient way to plot in *ggbio*, to create more customized graphics or to understand what happened inside `autoplot` function, you may want to create your own graphics layer by layer. In *ggbio*, generic function `ggplot` used to create plots by layers, it supports many core data objects defined in Bioconductor, it takes in the original data, and save it in `.data` element of the object, you can use `obj$.data` to get the original data, and a *data.frame* transformed and stored in the object too. Running `ggplot` function is just creating the **data** layer, no plot will be generated. You have to specify statistics and geometry by adding components using `+`.

For example, we can make some arches.

```
ggplot(gr) + geom_arch(aes(height = value))
```

```
## Object of class "ggbio"
```



Besides all components defined in *ggplot2*, we have several newly defined components inside *ggbio*. Let's take a look at a table about stat/geom/layout/coord/scale supported in *ggbio*,

> Tips: A good source for understanding the low level components is to read the on-line manual, they all parsed from example section from the Rd file. For *ggplot2*, it on `http://docs.ggplot2.org/current/`, for *ggbio* it's on `http://www.tengfei.name/ggbio/docs/man/`.

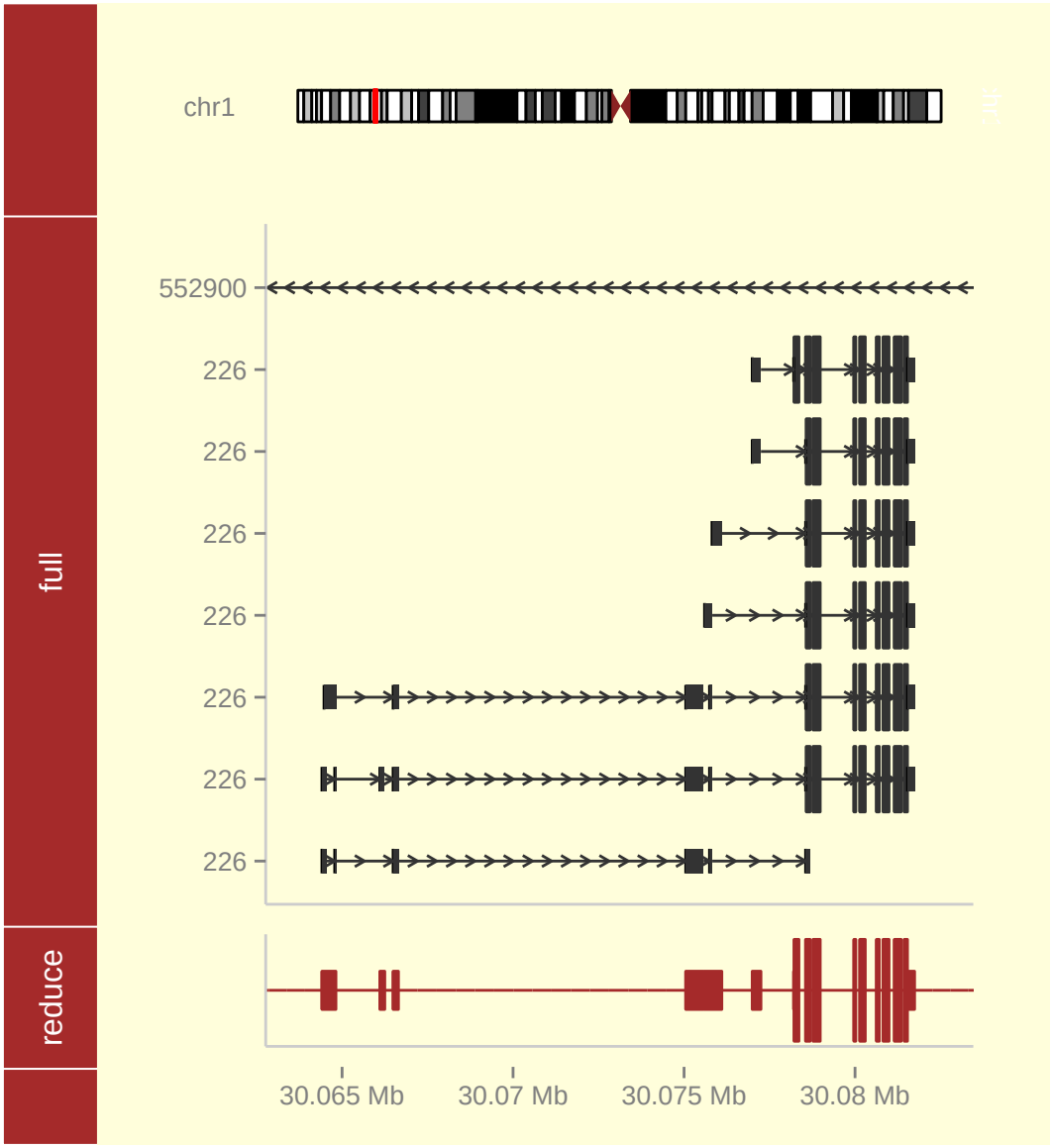
`plotIdeogram`(or `plotSingleChrom`) provides functionality to construct ideogram. `tracks` function provides convenient control to bind your individual graphics as tracks, reset/backup/modification is allowed.

```
library(ggbio)
## require internet connection
p.ideo <- plotIdeogram(genome = "hg19")
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
wh <- GRanges("chr16", IRanges(30064491, 30081734))
p1 <- autoplot(txdb, which = wh, names.expr = "gene_id")
```

Comp	name	usage	icon
geom	geom_rect	rectangle	
	geom_segment	segment	
	geom_chevron	chevron	
	geom_arrow	arrow	
	geom_arch	arches	
	geom_bar	bar	
	geom_alignment	alignment (gene)	
stat	stat_coverage	coverage (of reads)	
	stat_mismatch	mismatch pileup for alignments	
	stat_aggregate	aggregate in sliding window	
	stat_stepping	avoid overplotting	
	stat_gene	consider gene structure	
	stat_table	tabulate ranges	
	stat_identity	no change	
coord	linear	ggplot2 linear but facet by chromosome	
	genome	put everything on genomic coordinates	
	truncate gaps	compact view by shrinking gaps	
layout	track	stacked tracks	
	karyogram	karyogram display	
	circle	circular	
faceting	formula	facet by formula	
	ranges	facet by ranges	
scale	scale_x_sequnit	change x unit:Mb, kb, bp	
	scale_fill_giemsa	ideogram color	
	scale_fill_fold_change	around 0 scaling, for heatmap.	

Table 1.2: Components of the basic grammar of graphics, with the extensions available in *ggbio*.

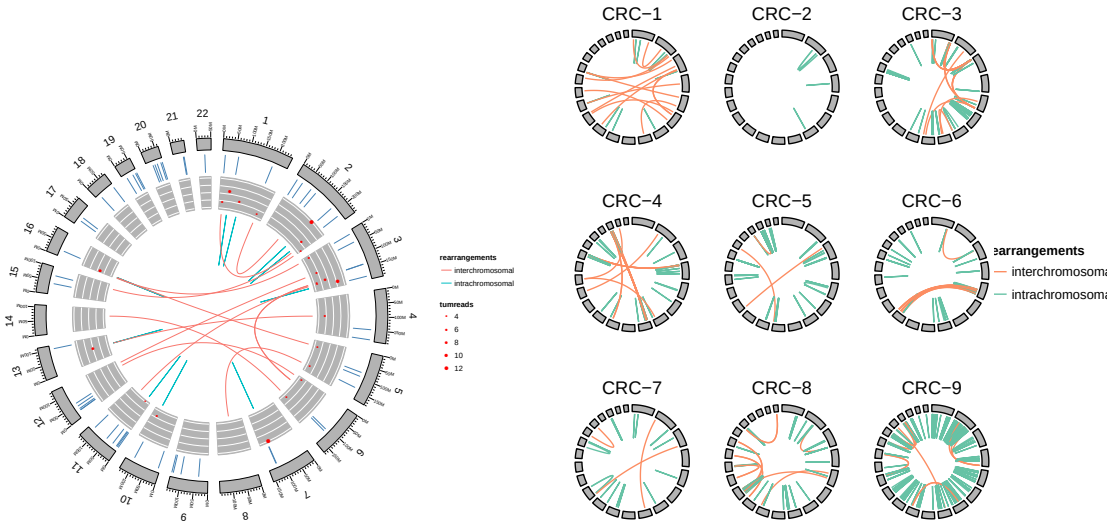
```
p2 <- autoplot(txdb, which = wh, stat = "reduce", color = "brown",
               fill = "brown")
tracks(p.ideo, full = p1, reduce = p2, heights = c(1.5, 5, 1)) +
  ylab("") + theme_tracks_sunset()
```



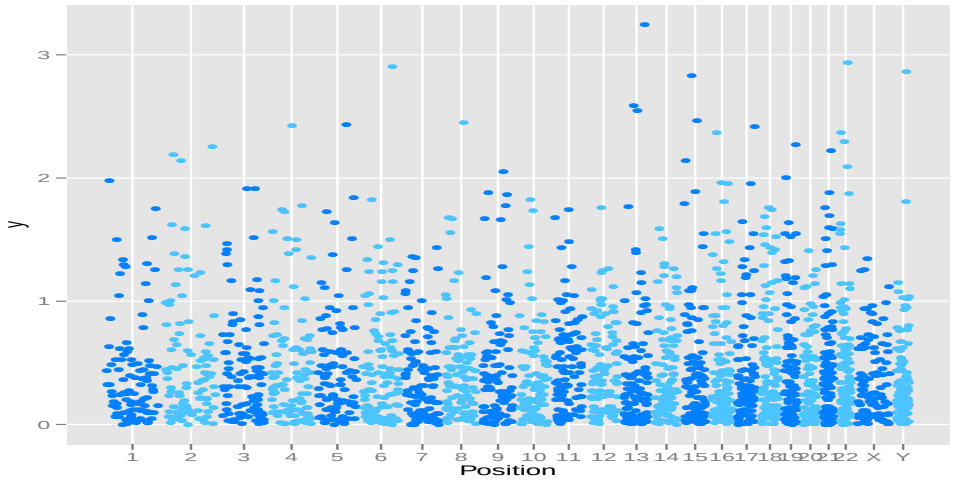
Manhattan plots are used to show SNP, circular view could be used to show chromosome rearrangement, kayrogram plot could be used to show clusterred events or observe distribution of haplotypes. In *ggbio*, `plotGrandLinear` is used to plot the whole genome Manhattan plot. Function `layout_karyogram` and layout 'karyogram' in `autoplot` to plot the karyogram overview. `layout_circle` and layout 'circle' in `autoplot` to plot the *GRanges* in circular layout.

If you are interested in how to visualize your data in circular layout like something shown in Figure

1.6.3, please go to chapter 6

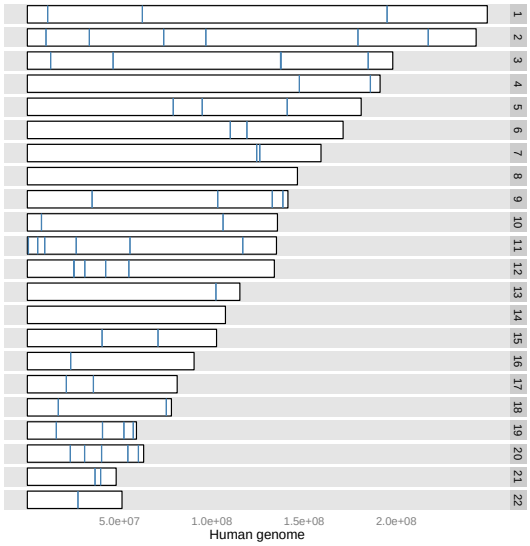


If you are interested in how to make manhattan plot like something shown in Figure 1.6.3, please go to chapter 7



If you are interested in how to visualize your data in karyogram layout like something shown in Figure 1.6.3, please go to chapter 8

For someother things like how to change theme and scales, please check chapter 14.


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
5.0e+07
1.0e+08
1.5e+08
2.0e+08
Human genome

Chapter 2

Visaulize gff-like files

For some historical reason, there are lots of different but very similar format out there to store interval data and meta data, for example, bed, gff, gtf, etc. Bioconductor provide very nice abstract for all kinds of widely used biological files. *GRanges* is one of them. With the help of pacakge *rtracklayer*, we can easily import files like *gff, bed* into R sesseion. For example

```
## fix annotation automatically
library(rtracklayer)
fl <- "~/Softwares/genome_browser/data/wgEncodeCshlLongRnaSeqHmecCellPamGeneDeNovoV2.gff"
gr <- import(, asRangedData = FALSE)
library(ggbio)
## fix me
autoplot(gr[seqnames(gr) == "chr1"], geom = "bar")
autoplot(gr[seqnames(gr) == "chr1"], geom = "bar", color = "black", aes(y = log(score)))
autoplot()
```

Chapter 3

Visualize bam files

```
fl <- "~/Datas/seqs/ENCODE/cshl/wgEncodeCshlLongRnaSeqGm12878CellPapAlnRep1.bam"
autoplot(fl)
p <- autoplot(fl, which = c(paste0("chr", 1:12)))
p + facet_wrap(~seqnames)
data(genesymbol, package = "biovizBase")
autoplot(fl, which = genesymbol["BRCA1"], method = "raw")
autoplot(fl, which = genesymbol["BRCA1"], method = "raw", geom = "area")
## fix me
autoplot(fl, which = genesymbol["BRCA1"], method = "raw", geom = "rect")
## fix me
autoplot(fl, which = genesymbol["BRCA1"], method = "raw", stat = "stepping")
autoplot(fl, which = genesymbol["BRCA1"], method = "raw", geom = "gapped.pair")
```

Chapter 4

How to make tracks

4.1 Motivation

`tracks` function could be used with **any other ggplot2 graphics**, not just for graphics produced by *ggbio*. *ggbio* depends on *ggplot2* and extends it to genomic world, **so most graphics produced by *ggbio* is essentially a *ggplot2* object, so you can use any tricks works for *ggplot2* on *ggbio* graphics.**.

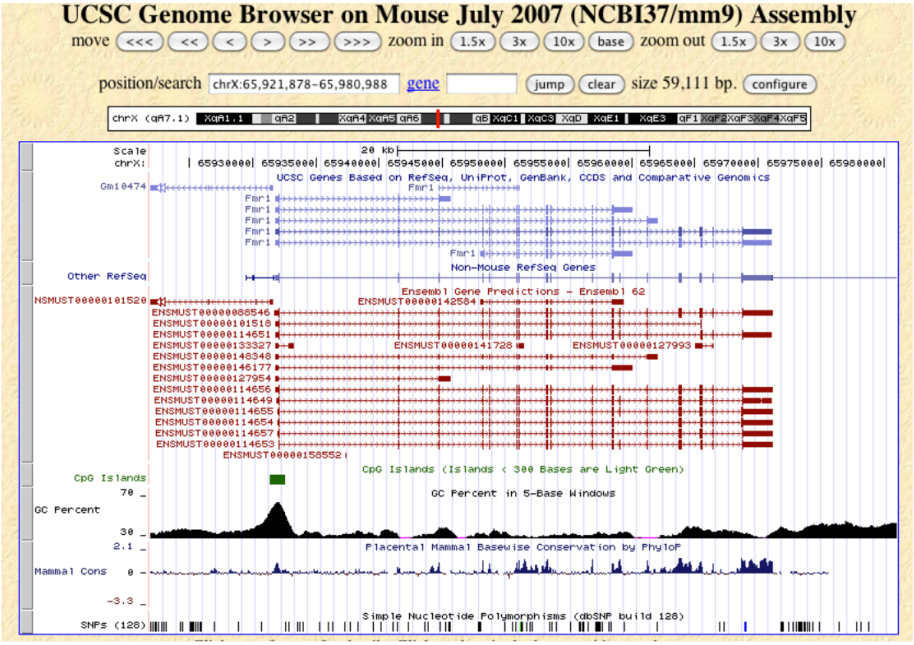
> Tips: If you want to manipulate graphics from *ggbio* more freely, documentation on *ggplot2* is a good start, *grid, gtable* packages are necessary knowledge for advanced users. Tracks relies on the new *gtable* package heavily, it has several convenient ways to manipulate the graphic objects.

Track-based view are widely used in almost all genome viewers, it usually stacks multiple plots row by row and align them on exactly the same coordinate, which in most cases, the genomic coordinates. In this way, we could be able to align various annotation data against each other to make comparison. UCSC genome browser[1] is one of the most widely used track-based web genome browser, as shown in Figure **??**. There are some other packages in R, that support track-based view like UCSC genome browser, such as *Gviz*.

ggbio is trying to be even more general in terms of building tracks, and offer more features.

- You can bind any graphics produced by *ggplot2*, not necessarily produced by *ggbio*, users could construct plots independently, and `tracks` will align them for you.
- Utilities for zooming, backup, restore a view. This is useful when you tweak around with your best snapshot, so you can always go back.

[1] `http://genome.ucsc.edu/cgi-bin/hgGateway`



- An extended + method. If you are familiar with *ggplot2*'s + method to edit an existing plot, this is the way it works, if tracks object is adding anything behind with + , that modification will be applied to each track. This make it easy to tweak with theme and update all the plots.
- Modify individual plot, with additional attributes, for example, 'fixed', 'mutable', etc . These attributes ONLY reflect when those plots are embeded into tracks function. Table 4.1 lists most attributes used.
- Creating your own customized themes for not only single plot but also tracks. We will show an example how to create a theme called `theme_tracks_subset` in the following sections.

attributes	description
bgColor	background color
fixed	fixed x scale or not
hline labeled	track is labeled on left or not
mutable	track is mutable to modification or not
hasAxis	track has x axis or not
height	height for track

Table 4.1: List of attributes, they all have corresponding replacer function such as RcodebgColor() ¡-

Tips: `tracks` function only support graphic objects produced by either *ggplot2* or *ggbio*. If you want to align plots, produced by other grid based system, like lattice, users need to tweak in grid level, to insert a lattice grob to a layout.

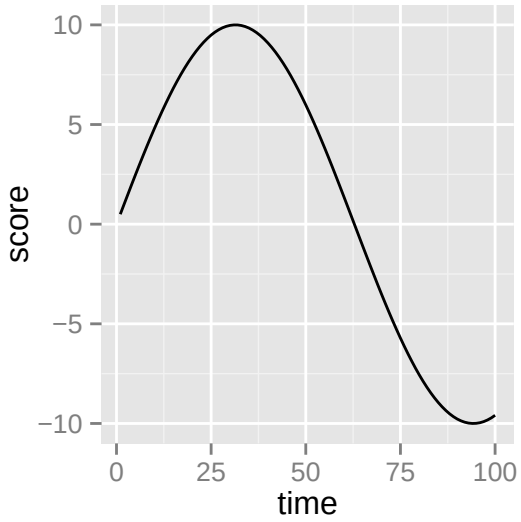
4.2 Minimal examples for `tracks`

Function **tracks** is a constructor for an object with class *Tracks*. This object is a container for each plot you are going to align, and all the graphic attributes controlling the appearance of tracks.

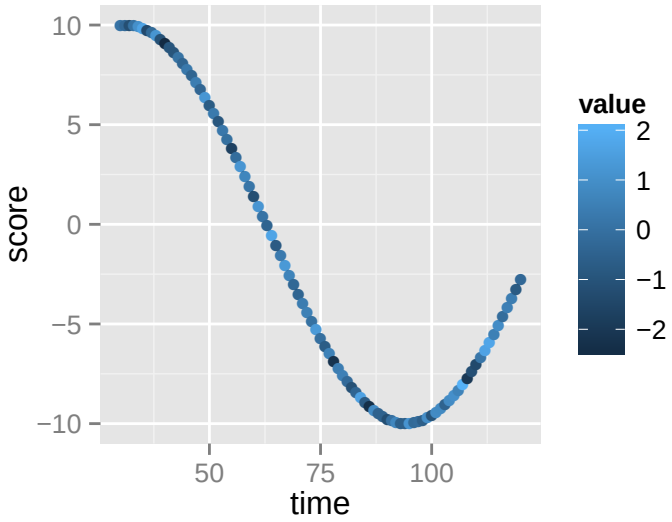
```
## load ggbio automatically load ggplot2
library(ggbio)
## make a simulated time series data set
df1 <- data.frame(time = 1:100, score = sin((1:100)/20)*10)
p1 <- qplot(data = df1, x = time, y = score, geom = "line")
df2 <- data.frame(time = 30:120, score = sin((30:120)/20)*10, value = rnorm(120-30 + 1))
p2 <- ggplot(data = df2, aes(x = time, y = score)) +
  geom_line() + geom_point(size = 2, aes(color = value))
```

Tips: When you see **qplot** function, you have to know it's *ggplot2*'s function(means 'quick plot'), since Bioconductor 2.10, *ggbio* stop using a confusing generic **qplot** function, instead, we are using a new generic method introduced in *ggplot2*, called **autoplot**.

```
p1
```

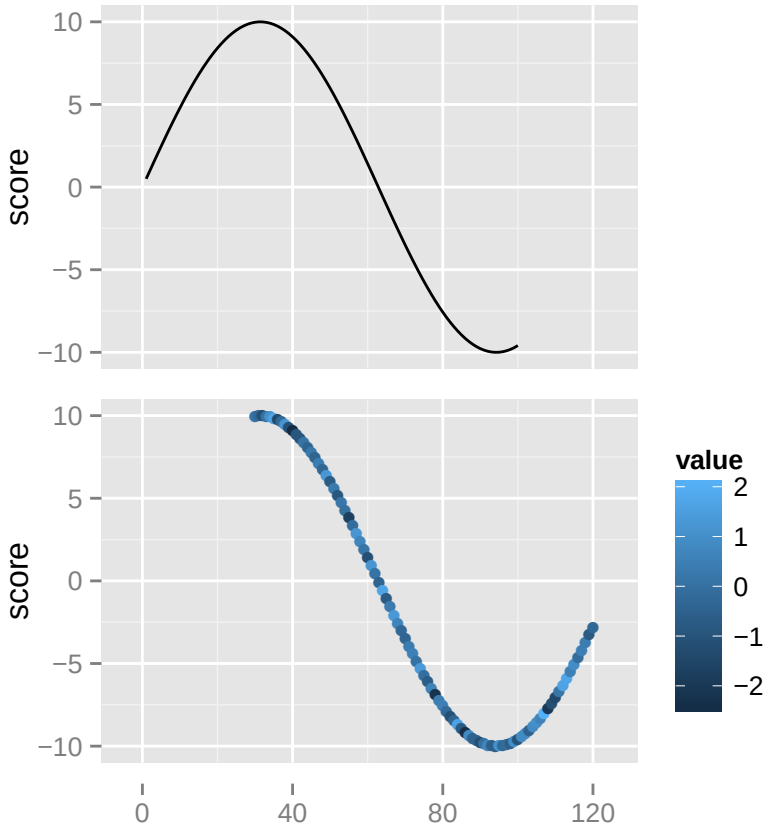



```
p2
```

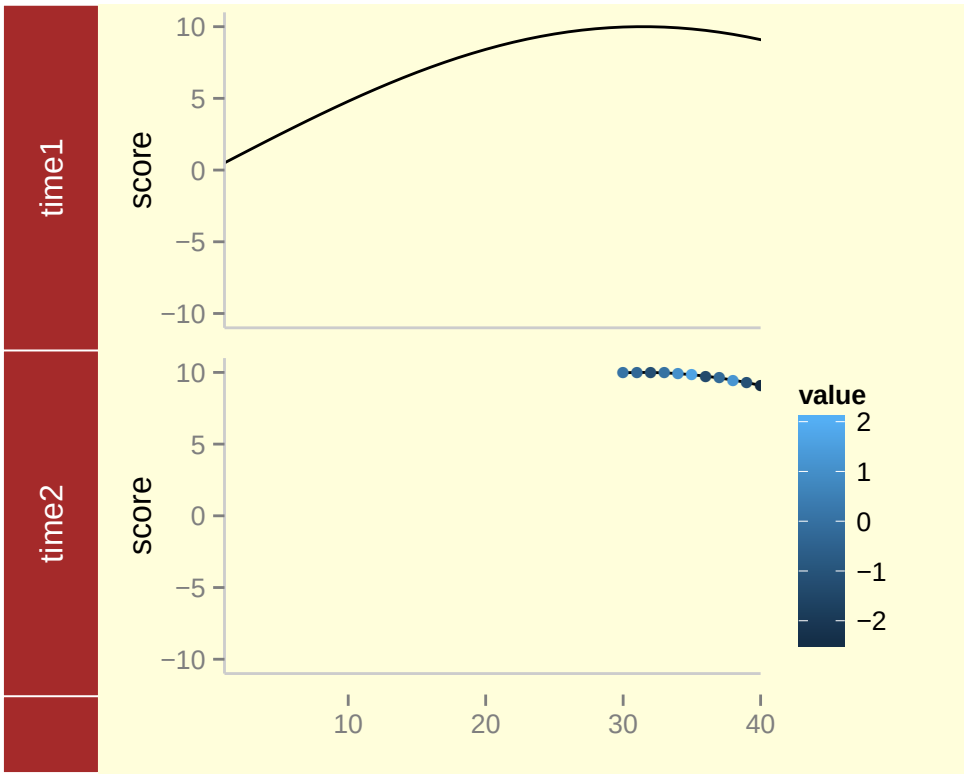


These two plots have different scale on x-axis, but we want to compare those two plots and hope to align them on exactly the same x-axis scale, then we could make vertical comaprison easily. By default, if you don't pass a name, the `tracks` simply align two plots without two labels. Notice even one plot has a legend, that won't affact the alignment.

```
tracks(p1, p2)
```




```
tracks(time1 = p1, time2 = p2) + xlim(1, 40) + theme_tracks_sunset()
```



Other availalbe zoom in/out methods:

```
library(GenomicRanges)
gr <- GRanges("chr", IRanges(1, 40))
# GRanges
tracks(time1 = p1, time2 = p2) + xlim(gr)
# IRanges
tracks(time1 = p1, time2 = p2) + xlim(ranges(gr))
tks <- tracks(time1 = p1, time2 = p2)
xlim(tks)
xlim(tks) <- c(1, 35)
xlim(tks) <- gr
xlim(tks) <- ranges(gr)
```

Check manual of `tracks` for other utilities like reset/backup.

Chapter 5

Visualize single chromosome

5.1 Introduction

Single Chromosome Ideogram: it is widely used in most track-based genome browsers, usually on top of all tracks, and use an indicator such as a highlighted window to indicate current zoomed region being viewed for different data tracks below, in this case, users won't lose too much context when zoomed into certain region.

We are going to introduce two types of single chromosome visualization in this vignette.

- The first one is used to be embedded into tracks as an overview, it's not a simple *ggplot* object. Only one highlighted rectangle are allowed to be plotted on top of it. We will focus mostly on this type of visualization in this vignette. In *ggbio*, this object belongs to a special class called 'ideogram', which has several effect which will be introduced later.
- If you want to render more data on single chromosome visualization, you have to use a special case for karyogram overview, which contains only one chromosome, more information about karyogram overview could be found in another vignettes about overview visualization.

5.2 Single chromosome visualization

5.2.1 Single chromosome use to be embedded in tracks.

`plotIdeogram` is a wrapper function around some functionality in package *rtracklayer* to help download cytoband table from UCSC automatically and return a graphic object with class 'ideogram'.

- If you don't pass genome name, it is going to ask your option from available genomes. NOTE: not all genome has cytoband information, if nocytoband information is available, only se-

qlengths information will be returned and a message will be printed. When *cytoband* information is available, the arm of chromosomes could be inferred, and plotted as you expected. You could always use `cytoband` argument to control it.

- If argument `subchr` is not specified, the first chromosomes is going to be used.

```
p <- plotIdeogram()
```

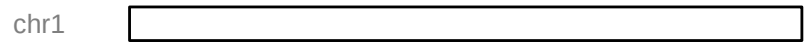
```
Please specify genome

  1: hg19       2: hg18       3: hg17       4: hg16       5: felCat4
  6: felCat3    7: galGal4    8: galGal3    9: galGal2   10: panTro3
 11: panTro2   12: panTro1   13: bosTau7   14: bosTau6   15: bosTau4
 16: bosTau3   17: bosTau2   18: canFam3   19: canFam2   20: canFam1
 21: loxAfr3   22: fr3       23: fr2       24: fr1       25: nomLeu1
 26: gorGor3   27: cavPor3   28: equCab2   29: equCab1   30: petMar1
 31: anoCar2   32: anoCar1   33: calJac3   34: calJac1   35: oryLat2
 36: myoLuc2   37: mm10      38: mm9       39: mm8       40: mm7
 41: hetGla1   42: monDom5   43: monDom4   44: monDom1   45: ponAbe2
 46: chrPic1   47: ailMel1   48: susScr2   49: ornAna1   50: oryCun2
 51: rn5       52: rn4       53: rn3       54: rheMac2   55: oviAri1
 56: gasAcu1   57: echTel1   58: tetNig2   59: tetNig1   60: melGal1
 61: macEug2   62: xenTro3   63: xenTro2   64: xenTro1   65: taeGut1
 66: danRer7   67: danRer6   68: danRer5   69: danRer4   70: danRer3
 71: ci2       72: ci1       73: braFlo1   74: strPur2   75: strPur1
 76: apiMel2   77: apiMel1   78: anoGam1   79: droAna2   80: droAna1
 81: droEre1   82: droGri1   83: dm3       84: dm2       85: dm1
 86: droMoj2   87: droMoj1   88: droPer1   89: dp3       90: dp2
 91: droSec1   92: droSim1   93: droVir2   94: droVir1   95: droYak2
 96: droYak1   97: caePb2    98: caePb1    99: cb3      100: cb1
101: ce10     102: ce6      103: ce4      104: ce2      105: caeJap1
106: caeRem3  107: caeRem2  108: priPac1  109: aplCal1  110: sacCer3
111: sacCer2  112: sacCer1

Selection:
```

After first plotting, the data is automatically hooked with the graphic object, when you do edit and zooming, it will NOT download it anymore, and you can even change the view to another chromosomes. That's the special part about object with class 'ideogram'.

```
library(ggbio)
## requrie connection
p <- plotIdeogram(genome = "hg19")
p
```

```
## Object of class "ideogram"
```


```
p <- plotIdeogram(genome = "hg19", cytoband = FALSE)
p

## Object of class "ideogram"
```




```
## the data stored with p, won't download again for zooming
head(attr(p, "ideogram.data"))

## GRanges with 6 ranges and 0 metadata columns:
##       seqnames          ranges strand
##          <Rle>       <IRanges>  <Rle>
##   [1]     chr1 [1, 249250621]      *
##   [2]     chr2 [1, 243199373]      *
##   [3]     chr3 [1, 198022430]      *
##   [4]     chr4 [1, 191154276]      *
##   [5]     chr5 [1, 180915260]      *
##   [6]     chr6 [1, 171115067]      *
```

```
##   ---
##   seqlengths:
##                     chr1                  chr2 ... chr18_gl000207_random
##                249250621             243199373 ...                  4262
```

> Tips: `aspect.ratio` by default is 1/20, if you set it to `NULL`, you have to resize the graphic device manually. You can always set the aspect.ration in theme() function of *ggplot2* by `+theme(aspect.ratio = )`

You can always download the data manualy and save it and use it later, the function used called `getIdoegram` in package *biovizBase*. Or more flexible relevant function in package *rtracklayer*. The data *hg19IdeogramCyto* is a default data of human in *ggbio*. What if you cannot get cytoband information from UCSC, but have the data available in hand? You can construct the `GRanges` object manually, but have to satisfy following restriction:

Object have to has elementMeta columns:

- name: start with p or q. to tell the different arms of chromosomes. such as **p36.22** and **q12**.
- gieStain: dye color of cytoband. such as **gneg**.

```
data(hg19IdeogramCyto, package = "biovizBase")
## data structure
head(hg19IdeogramCyto)


## GRanges with 6 ranges and 2 metadata columns:
##       seqnames               ranges strand |     name gieStain
##          <Rle>            <IRanges>  <Rle> | <factor> <factor>
##   [1]     chr1 [       0,  2300000]      * |   p36.33     gneg
##   [2]     chr1 [ 2300000,  5400000]      * |   p36.32   gpos25
##   [3]     chr1 [ 5400000,  7200000]      * |   p36.31     gneg
##   [4]     chr1 [ 7200000,  9200000]      * |   p36.23   gpos25
##   [5]     chr1 [ 9200000, 12700000]      * |   p36.22     gneg
##   [6]     chr1 [12700000, 16200000]      * |   p36.21   gpos50
##   ---
##   seqlengths:
##     chr1 chr10 chr11 chr12 chr13 chr14 ...  chr7  chr8  chr9  chrX  chrY
##       NA    NA    NA    NA    NA    NA ...    NA    NA    NA    NA    NA


plotIdeogram(hg19IdeogramCyto)


## Object of class "ideogram"
```

Here comes more special features about the single chromosome 'ideogram' object, it all aims to be conventient when it's embeded in tracks. For a normal *ggbio* plot or *ggplot2* plot object, when you set limmits, it zooms in certain ranges. But, for an 'ideogram' object, set limits will **only** add highlights rectangle!

You could specify argument `zoom.region` in `plotIdeogram` function, or plus a function `xlim`, it accpets

- nuemric range
- IRanges
- GRanges object, when it's GRanges object, it will change the chromosome if seqnames is not what it is before.

The highlighted style will be remembered when you zoom use xlim.

```
plotIdeogram(hg19IdeogramCyto, "chr1", zoom.region = c(1e7, 5e7))


## Object of class "ideogram"
```

```
## change style of highlighted rectangle
## p <- plotIdeogram(hg19IdeogramCyto, "chr1")
p <- plotIdeogram(hg19IdeogramCyto, "chr1",
             zoom.region = c(1e7, 5e7), fill = NA, color = "blue", size = 2,
             zoom.offset = 4)
p


## Object of class "ideogram"
```


```
class(p)


## [1] "ideogram"
## attr(,"package")
## [1] "ggbio"

p + xlim(1e7, 5e7)


## Object of class "ideogram"
```

```
library(GenomicRanges)
p + xlim(IRanges(5e7, 7e7))


## Object of class "ideogram"
```


```
## change visualized chromosomes
p + xlim(GRanges("chr2", IRanges(1e7, 5e7)))


## Object of class "ideogram"
```

Default ideogram has no X-scale label, to add axis text, you have to specify argument `xlabel` to `TRUE`.

```
plotIdeogram(hg19IdeogramCyto, "chr1", xlabel = TRUE)


## Object of class "ideogram"
```

Some time, you don't want to visualize a chromosome with cytobands, or you cannot find any information about cytobands, in this case, you can simply visualize a blank chromosome as overview template. *ggbio* has several ways to do it.

- Use argument `cytoband`. Set it to `FALSE`.
- Pass a GRanges with no extra column such as **name, gieStain**. it will automatically parse and estimate the chromosome lengths. It is **IMPORTANT** that to create an accurate lengths for chromosomes, you need to either make sure the ranges you passed covers all chromosomes or you need to specify the `seqlengths` for our `GRanges` object.
- Use autoplot,Seqinfo, when you only pass one chromosomes, it automatically convert it to an 'ideogram'.

When there is no seqlengths, the length is estiamted from the data(cytoband).

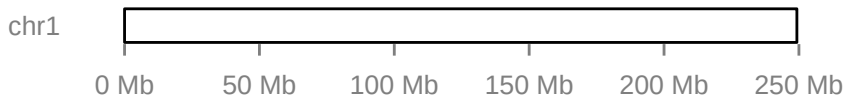
```
## there are no seqlengths
data(hg19IdeogramCyto, package = "biovizBase")
seqlengths(hg19IdeogramCyto)
```

```
##  chr1 chr10 chr11 chr12 chr13 chr14 chr15 chr16 chr17 chr18 chr19  chr2
##    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## chr20 chr21 chr22  chr3  chr4  chr5  chr6  chr7  chr8  chr9  chrX  chrY
##    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
```

```
## so directly plot will try to aggregate and estimate lengths of chromosomes,
## this is not accurate
p1 <- plotIdeogram(hg19IdeogramCyto, "chr1", cytoband = FALSE, xlabel = TRUE)
```

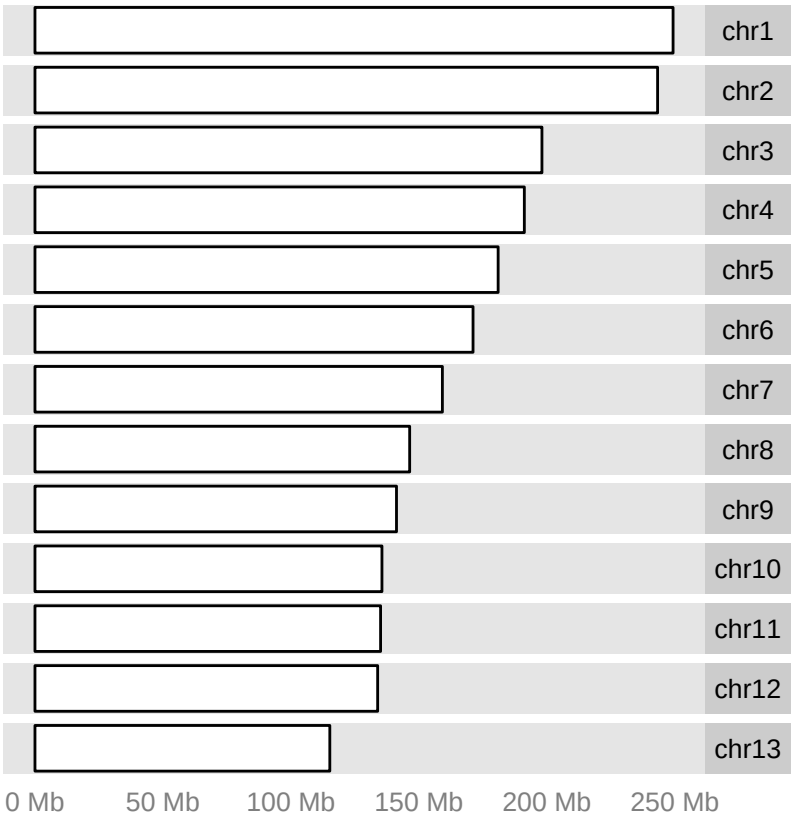
```
## Warning:  geom(ideogram) need valid seqlengths information for accurate mapping,
##                  now use reduced information as ideogram...
```

```
p1


## Object of class "ideogram"
```



Another default data 'hg19Ideogram' contains seqlengths, more suitable for plotting blank overview. Use 'Seqinfo' is convenient way to construct single chromosome overview or karyogram overview.

```
data(hg19Ideogram, package= "biovizBase")
autoplot(seqinfo(hg19Ideogram)[paste0("chr", 1:13)])
```

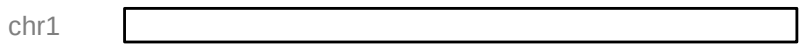


Single chromosome visualization by seqinfo, if argument `ideogram` is set to TRUE, the returned object is an 'ideogram' object. By default, it's a normal ggplot object, their lookings are different too.

```
head(hg19Ideogram)


## GRanges with 6 ranges and 0 metadata columns:
##                    seqnames         ranges strand
##                       <Rle>      <IRanges>  <Rle>
##   [1]                 chr1 [1, 249250621]      *
##   [2] chr1_gl000191_random [1,    106433]      *
##   [3] chr1_gl000192_random [1,    547496]      *
##   [4]                 chr2 [1, 243199373]      *
##   [5]                 chr3 [1, 198022430]      *
##   [6]                 chr4 [1, 191154276]      *
##   ---
##   seqlengths:
##                     chr1  chr1_gl000191_random ...                 chrM
##                249250621                106433 ...                16571
```

```
library(GenomicRanges)
## single ideogram
p <- autoplot(seqinfo(hg19Ideogram)["chr1"])
p + theme(aspect.ratio = 1/20)
```


```
class(p)


## [1] "gg"     "ggplot"


p <- autoplot(seqinfo(hg19Ideogram)["chr1"], ideogram = TRUE)
p


## Object of class "ideogram"
```




```
class(p)


## [1] "ideogram"
## attr(,"package")
## [1] "ggbio"
```

To add more data freely on your single chromosome overview, I can see cases that users are familiar with *ggbio* and *ggplot2* and they hope to

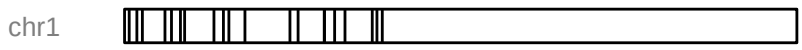
- Tweak with graphics more before embedded in tracks.
- Just visualize data on a single chromosome.

You can

- Set `ideogram` to `TRUE`, and change class back to ggplot default, then tweak with low level function.
- Default then use `layout_karyogram`.

use argument `ideogram` to set it to `FALSE`, then it's just a formal ggplot object, and you could manipulate it as usual.

```
## not ideogram, just ggplot object
p <- autoplot(seqinfo(hg19Ideogram)["chr1"], ideogram = TRUE)
class(p)


## [1] "ideogram"
## attr(,"package")
## [1] "ggbio"


class(p) <- c("gg", "ggplot")


## Warning:  Setting class(x) to multiple strings ("gg", "ggplot", ...); result will
no longer be an S4 object


gr <- GRanges("chr1", IRanges(start = sample(1:1e8, size = 20), width = 5),
  seqlengths = seqlengths(hg19Ideogram)["chr1"])
library(biovizBase)
p + geom_rect(data = mold(gr), aes(xmin = start, xmax = start, ymin = 0, ymax = 10),
              fill = "black", color = "black")
```




```
## or default + layout_karyogram
p <- autoplot(seqinfo(hg19Ideogram)["chr1"]) + layout_karyogram(gr) + theme(aspect.ratio = 1/
p
```

5.2.2 Get ideogram or customize the colors

We only provide default cytoband ideogram information and trying to cover all the cases might be encountered in real world, but what if you want to create your ideogram color yourself? To update the cytoband color with complete definition, simply replace the pre-defined color set. This will affect all the R session.

```
optlist <- getOption("biovizBase")
cyto.new <- rep(c("red", "blue"), length = length(optlist$cytobandColor))
names(cyto.new) <- names(optlist$cytobandColor)
head(cyto.new)


##   gneg  stalk   acen   gpos   gvar  gpos1
##  "red" "blue"  "red" "blue"  "red" "blue"


## suppose cyto.new is your new defined color
```

```
optlist$cytobandColor <- cyto.new
options(biovizBase = optlist)
## see what happenned...
plotIdeogram(hg19IdeogramCyto)

## Object of class "ideogram"
```

Chapter 6

Circular view

6.1 Introduction

Circular view is a special layout in *ggbio* , this idea has been implemented in many different software, for example, the Circos project.

In this tutorial, we will start from the raw data, if you are already familiar with how to process your data into the right format, which here I mean `GRanges`,you can jump to 6.2.3 directly.

6.2 Tutorial

6.2.1 Step 1: understand the layout circle

We have discussed about the new coordinate "genome" in vignette about Manhattan plot before, now this time, it's one step further compared to genome coordinate transformation. We specify ring radius `radius` and track width `trackWidth` to help transform a linear genome coordinate system to a circular coordinate system. By using `layout_circle` function which we will introduce later.

Before we visualize our data, we need to have something in mind

- How many tracks we want?
- Can they be combined into the same data?
- Do I have chromosomes lengths information?
- Do I have interesting variables attached as one column?

6.2.2 Step 2: get your data ready to plot

Ok, let's start to process some raw data to the format we want. The data used in this study is from this a paper[1]. In this example, We are going to

1. Visualize somatic mutation as segment.
2. Visualize inter,intro-chromosome rearrangement as links.
3. Visualize mutation score as point tracks with grid-background.
4. Add scale and ticks and labels.
5. To arrange multiple plots and legend. create multiple sample comparison.

Notes: don't put too much tracks on it.

I simply put script here to get mutation data as 'GRanges' object.

```
crc1 <- system.file("extdata", "crc1-missense.csv", package = "biovizBase")
crc1 <- read.csv(crc1)
library(GenomicRanges)
mut.gr <- with(crc1,GRanges(Chromosome, IRanges(Start_position, End_position),
                            strand = Strand))
values(mut.gr) <- subset(crc1, select = -c(Start_position, End_position, Chromosome))
data("hg19Ideogram", package = "biovizBase")
seqs <- seqlengths(hg19Ideogram)
## subset_chr
chr.sub <- paste("chr", 1:22, sep = "")
## levels tweak
seqlevels(mut.gr) <- c(chr.sub, "chrX")
mut.gr <- keepSeqlevels(mut.gr, chr.sub)
seqs.sub <- seqs[chr.sub]
## remove wrong position
bidx <- end(mut.gr) <= seqs.sub[match(as.character(seqnames(mut.gr)),
              names(seqs.sub))]
mut.gr <- mut.gr[which(bidx)]
## assign_seqlengths
seqlengths(mut.gr) <- seqs.sub
## reanme to shorter names
new.names <- as.character(1:22)
names(new.names) <- paste("chr", new.names, sep = "")
new.names
```

[1] http://www.nature.com/ng/journal/v43/n10/full/ng.936.html

```
##  chr1  chr2  chr3  chr4  chr5  chr6  chr7  chr8  chr9 chr10 chr11 chr12
##   "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"  "10"  "11"  "12"
## chr13 chr14 chr15 chr16 chr17 chr18 chr19 chr20 chr21 chr22
##  "13"  "14"  "15"  "16"  "17"  "18"  "19"  "20"  "21"  "22"

mut.gr.new <- renameSeqlevels(mut.gr, new.names)
head(mut.gr.new)

## GRanges with 6 ranges and 10 metadata columns:
##       seqnames                 ranges strand | Hugo_Symbol
##          <Rle>              <IRanges>  <Rle> |    <factor>
##   [1]        1 [ 11003085,  11003085]      + |      TARDBP
##   [2]        1 [ 62352395,  62352395]      + |       INADL
##   [3]        1 [194960885, 194960885]      + |         CFH
##   [4]        2 [ 10116508,  10116508]      - |        CYS1
##   [5]        2 [ 33617747,  33617747]      + |     RASGRP3
##   [6]        2 [ 73894280,  73894280]      + |     C2orf78
##       Entrez_Gene_Id   Center NCBI_Build   Strand
##            <integer> <factor>  <integer> <factor>
##   [1]          23435    Broad         36        +
##   [2]          10207    Broad         36        +
##   [3]           3075    Broad         36        +
##   [4]         192668    Broad         36        -
##   [5]          25780    Broad         36        +
##   [6]         388960    Broad         36        +
##       Variant_Classification Variant_Type Reference_Allele
##                     <factor>     <factor>         <factor>
##   [1]              Missense          SNP                G
##   [2]              Missense          SNP                T
##   [3]              Missense          SNP                G
##   [4]              Missense          SNP                C
##   [5]              Missense          SNP                C
##   [6]              Missense          SNP                T
##       Tumor_Seq_Allele1 Tumor_Seq_Allele2
##                <factor>         <factor>
##   [1]                 G                A
##   [2]                 T                G
##   [3]                 G                A
##   [4]                 C                T
##   [5]                 C                T
##   [6]                 T                C
##   ---
##   seqlengths:
##           1         2         3 ...        20        21        22
```

```
##    249250621 243199373 198022430 ...  63025520  48129895  51304566
```

To get ideogram track, we need to load human hg19 ideogram data, for details please check another vignette about getting ideogram.

```
hg19Ideo <- hg19Ideogram
hg19Ideo <- keepSeqlevels(hg19Ideogram, chr.sub)
hg19Ideo <- renameSeqlevels(hg19Ideo, new.names)
head(hg19Ideo)


## GRanges with 6 ranges and 0 metadata columns:
##       seqnames         ranges strand
##          <Rle>      <IRanges>  <Rle>
##   [1]        1 [1, 249250621]      *
##   [2]        2 [1, 243199373]      *
##   [3]        3 [1, 198022430]      *
##   [4]        4 [1, 191154276]      *
##   [5]        5 [1, 180915260]      *
##   [6]        6 [1, 171115067]      *
##   ---
##   seqlengths:
##           1         2         3 ...        20        21        22
##    249250621 243199373 198022430 ...  63025520  48129895  51304566
```

6.2.3 Step 3: low level API: `layout_circle`

`layout_circle` is a lower level API for creating circular plot, it accepts `Granges` object, and users need to specify radius, track width, and other aesthetics, it's very flexible. But keep in mind, you **have to** pay attention rules when you make circular plots.

- For now, `seqlengths`, `seqlevels` and chromosomes names should be exactly the same, so you have to make sure data on all tracks have this uniform information to make a comparison.
- Set arguments `space.skip` to the same value for all tracks, that matters for transformation, default is the same, so you don't have to change it, unless you want to add/remove space in between.
- `direction` argument should be exactly the same, either "clockwise" or "counterclockwise".
- Tweak with your radius and tracks width to get best results.

Since low level API leave you as much flexibility as possible, this may looks hard to adjust, but it can produce various types of graphics which higher levels API like `autoplot` hardly can, for instance, if you want to overlap multiple tracks or fine-tune your layout.

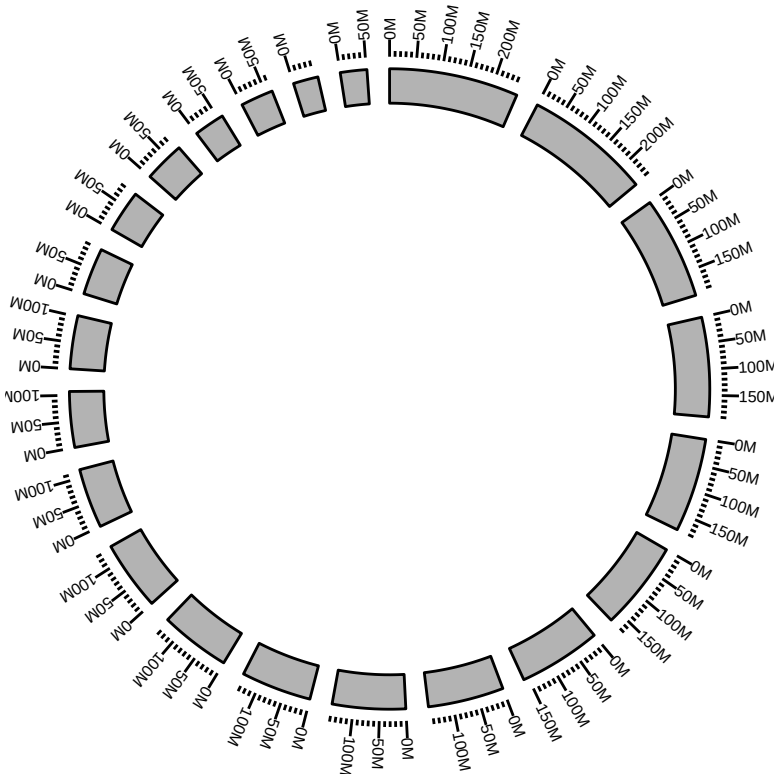
Ok, let's start to add tracks one by one.

First to add a "ideo" track

```
library(ggbio)
p <- ggplot() + layout_circle(hg19Ideo, geom = "ideo", fill = "gray70",
                              radius = 30, trackWidth = 4)
```

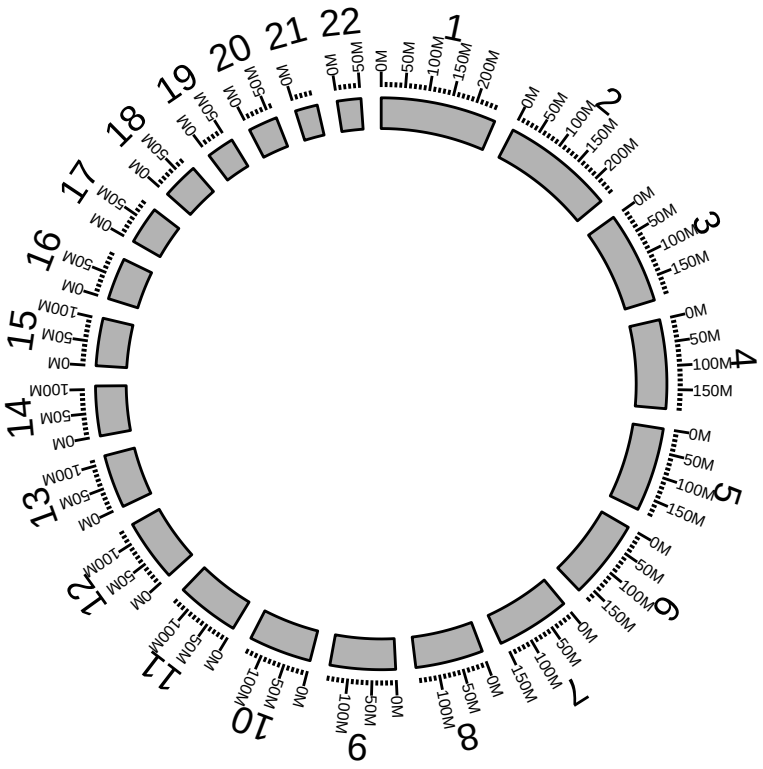
Then a "scale" track with ticks

```
p <- p + layout_circle(hg19Ideo, geom = "scale", size = 2, radius = 35, trackWidth = 2)
p
```



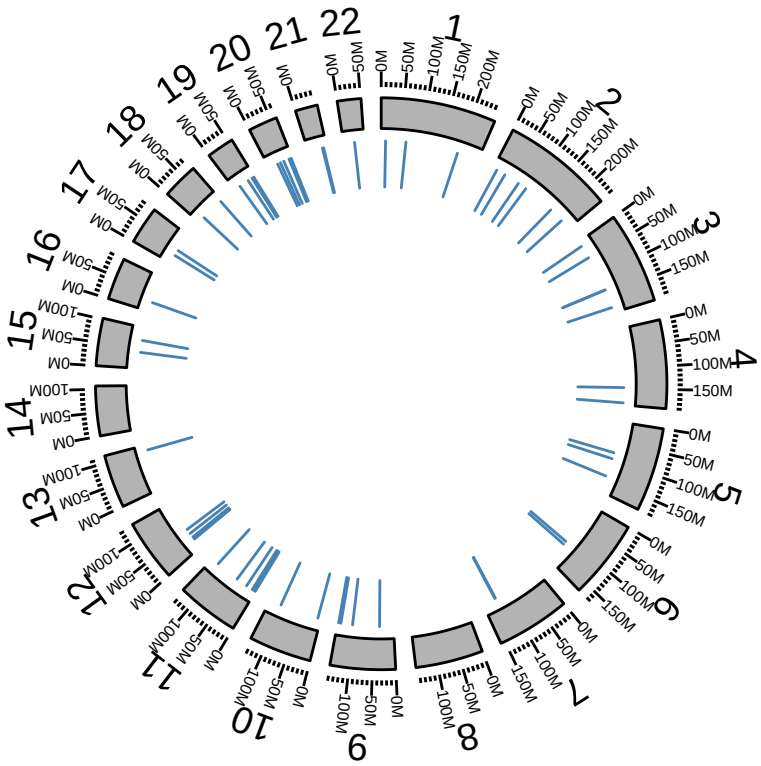
Then a "text" track to label chromosomes. *NOTICE*, after genome coordinate transformation, original data will be stored in column ".ori", and for mapping, just use ".ori" prefix to it. Here we use '.ori.seqnames', if you use 'seqnames', that is going to be just "genome" character.

```
p <- p + layout_circle(hg19Ideo, geom = "text", aes(label = seqnames), vjust = 0,
                       radius = 38, trackWidth = 7)
p
```



Then a "rectangle" track to show somatic mutation, this will looks like vertical segments.

```
p <- p + layout_circle(mut.gr, geom = "rect", color = "steelblue",
                       radius = 23 ,trackWidth = 6)
p
```



Next, we need to add some "links" to show the rearrangement, of course, links can be used to map any kind of association between two or more different locations to indicate relationships like copies or fusions.

```
rearr  <- read.csv(system.file("extdata", "crc-rearrangment.csv", package = "biovizBase"))
## start position
gr1 <- with(rearr, GRanges(chr1, IRanges(pos1, width = 1)))
## end position
gr2 <- with(rearr, GRanges(chr2, IRanges(pos2, width = 1)))
## add extra column
nms <- colnames(rearr)
.extra.nms <- setdiff(nms, c("chr1", "chr2", "pos1", "pos2"))
values(gr1) <- rearr[,.extra.nms]
## remove out-of-limits data
seqs <- as.character(seqnames(gr1))
.mx <- seqlengths(hg19Ideo)[seqs]
idx1 <- start(gr1) > .mx
seqs <- as.character(seqnames(gr2))
.mx <- seqlengths(hg19Ideo)[seqs]
idx2 <- start(gr2) > .mx
```

```
idx <- !idx1 & !idx2
gr1 <- gr1[idx]
seqlengths(gr1) <- seqlengths(hg19Ideo)
gr2 <- gr2[idx]
seqlengths(gr2) <- seqlengths(hg19Ideo)
```

To create a suitable structure to plot, please use another ‘GRanges‘ to represent the end of the links, and stored as elementMetadata for the ”start point” ‘GRanges‘. Here we named it as ”to.gr” and will be used later.

```
values(gr1)$to.gr <- gr2
## rename to gr
gr <- gr1
```

Here we show the flexibility of *ggbio*, for example, if you want to use color to indicate your links, make sure you add extra information in the data, used for mapping later. Here in this example, we use ”intrachromosomal” to label rearrangement within the same chromosomes and use ”interchromosomal” to label rearrangement in different chromosomes.

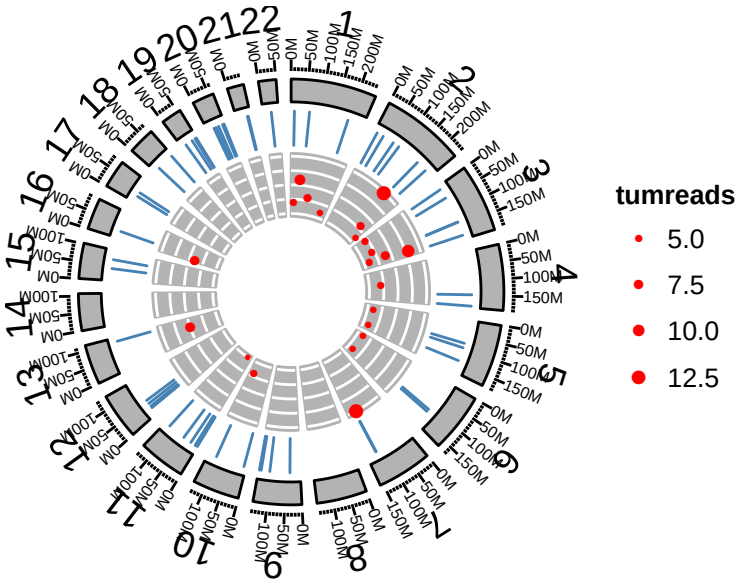
```
values(gr)$rearrangements <- ifelse(as.character(seqnames(gr))
                                    == as.character(seqnames((values(gr)$to.gr))),
                                    "intrachromosomal", "interchromosomal")
```

Get subset of links data for only one sample ”CRC1”

```
gr.crc1 <- gr[values(gr)$individual == "CRC-1"]
```

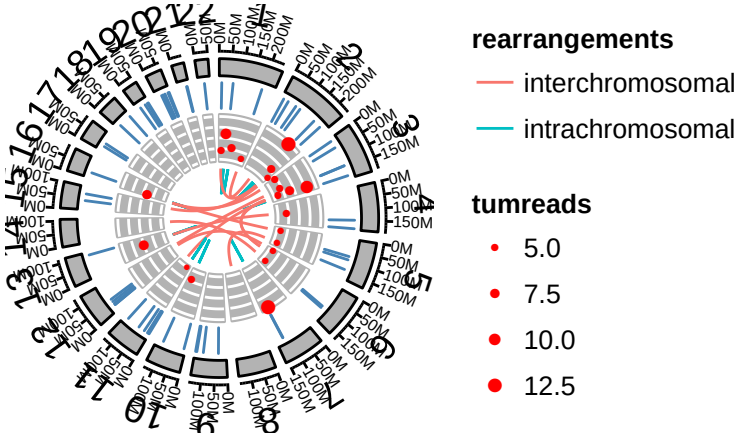
Ok, add a ”point” track with grid background for rearrangement data and map ‘y‘ to variable ”score”, map ‘size‘ to variable ”tumreads”, rescale the size to a proper size range.

```
p <- p + layout_circle(gr.crc1, geom = "point", aes(y = score, size = tumreads), color = "red
                       radius = 12 ,trackWidth = 10, grid = TRUE) +
  scale_size(range = c(1, 2.5))
p
```



Finally, let's add links and map color to rearrangement types. Remember you need to specify 'linked.to' to the column that contain end point of the data.

```
p <- p + layout_circle(gr.crc1, geom = "link", linked.to = "to.gr", aes(color = rearrangement
                       radius = 10 ,trackWidth = 1)
p
```

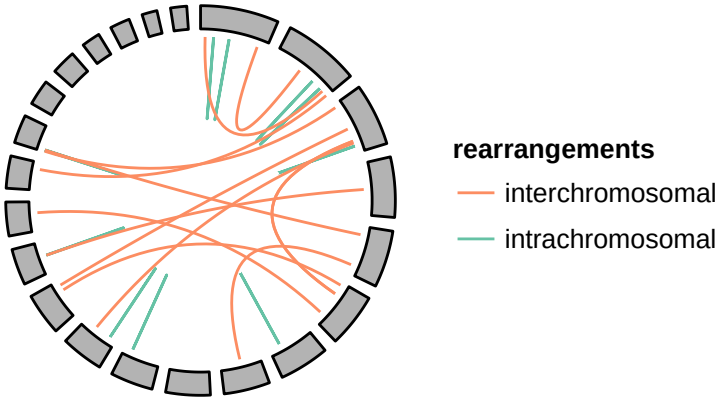


6.2.4 Step 4: Complex arragnment of plots

In this step, we are going to make multiple sample comparison, this may require some knowledge about package *grid* and *gridExtra*. We will introduce a more easy way to combine your graphics later after this.

We just want 9 single circular plots put together in one page, since we cannot keep too many tracks, we only keep ideogram and links. Here is one sample.

```
cols <- RColorBrewer::brewer.pal(3, "Set2")[2:1]
names(cols) <- c("interchromosomal", "intrachromosomal")
p0 <- ggplot() + layout_circle(gr.crc1, geom = "link", linked.to = "to.gr",
                               aes(color = rearrangements), radius = 7.1) +
  layout_circle(hg19Ideo, geom = "ideo", trackWidth = 1.5,
                color = "gray70", fill = "gray70") +
  scale_color_manual(values = cols)
p0
```



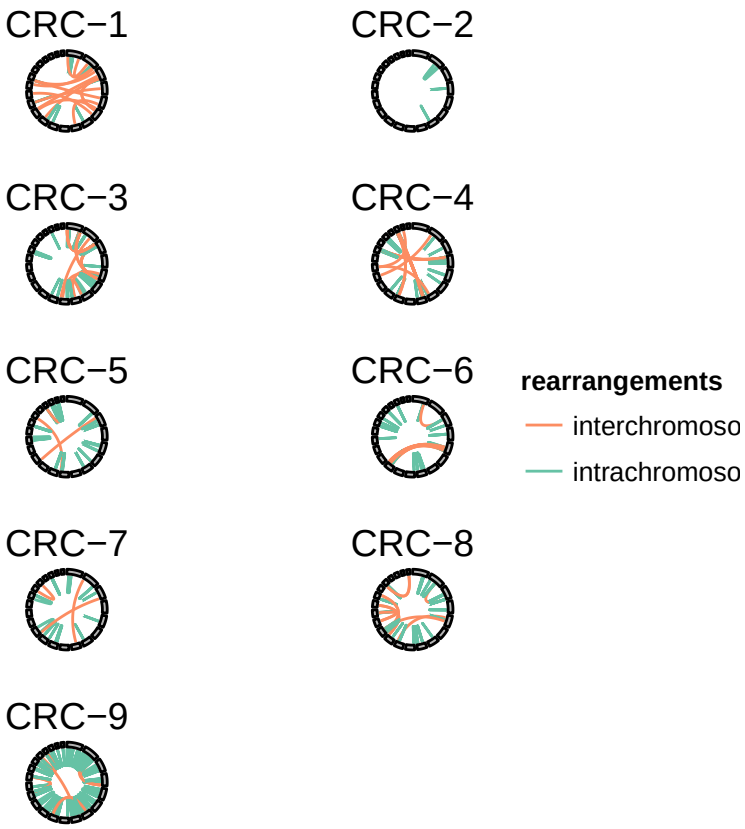

```
grl <- split(gr, values(gr)$individual)
## need "unit", load grid
library(grid)
lst <- lapply(grl, function(gr.cur){
  print(unique(as.character(values(gr.cur)$individual)))
  cols <- RColorBrewer::brewer.pal(3, "Set2")[2:1]
  names(cols) <- c("interchromosomal", "intrachromosomal")
  p <- ggplot() + layout_circle(gr.cur, geom = "link", linked.to = "to.gr",
                          aes(color = rearrangements), radius = 7.1) +
                            layout_circle(hg19Ideo, geom = "ideo", trackWidth = 1.5,
                          color = "gray70", fill = "gray70") +
                            scale_color_manual(values = cols)  +
     labs(title = (unique(values(gr.cur)$individual))) +
       theme(plot.margin = unit(rep(0, 4), "lines"))
})
```

```
## [1] "CRC-1"
## [1] "CRC-2"
## [1] "CRC-3"
## [1] "CRC-4"
```

```
## [1] "CRC-5"
## [1] "CRC-6"
## [1] "CRC-7"
## [1] "CRC-8"
## [1] "CRC-9"
```

We wrap the function in grid level to a more user-friendly high level function, called `arrangeGrobByParsingLegen` You can pass your ggplot2 graphics to this function , specify the legend you want to keep on the right, you can also specify the column/row numbers. Here we assume all plots we have passed follows the same color scale and have the same legend, so we only have to keep one legend on the right.

```
arrangeGrobByParsingLegend(lst, widths = c(4, 1), legend.idx = 1, ncol = 2)
```

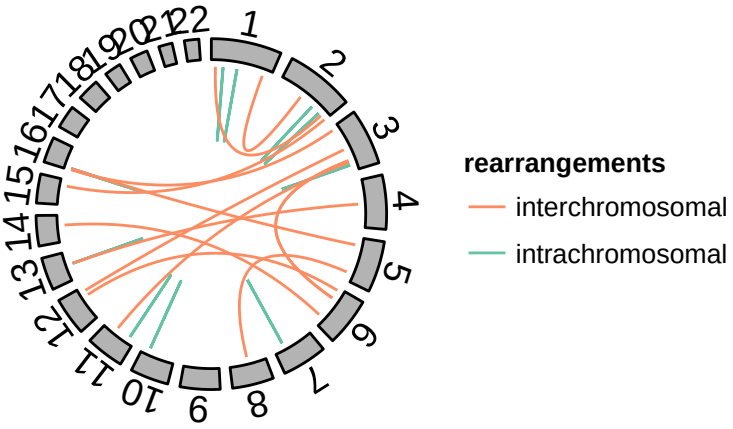


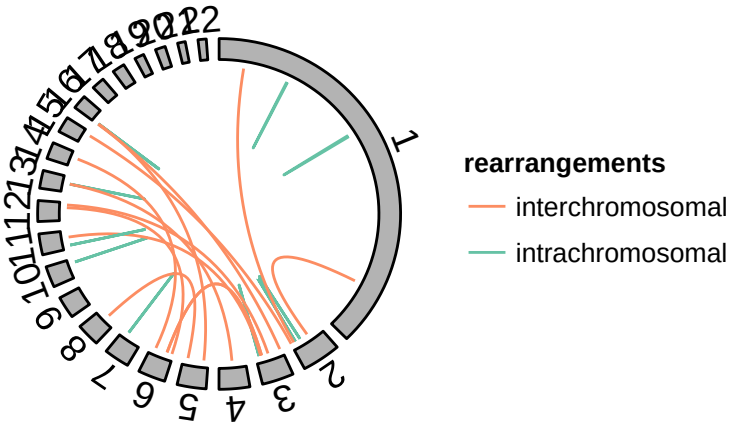

```
## NULL
```

6.3 Transform space

This is an experimental feature that added after 1.7.12, which transform the genome space based on some specified proportions. In `layout_circle` there is a new parameter called `chr.weight`, which is a vector of numeric value and sum of those value should not exceed 1, these value indicates proportion of chrommosome space to take in overall space. Names of this vectors are chromosomes names, and you can only specify a few of them, other chromosomes will take up left space according to their space.

```
p1 <- ggplot() + layout_circle(gr.crc1, geom = "link", linked.to = "to.gr",
                               aes(color = rearrangements), radius = 7.1) +
  layout_circle(hg19Ideo, geom = "ideo", trackWidth = 1.5,
                color = "gray70", fill = "gray70") +
  layout_circle(hg19Ideo, geom = "text", trackWidth = 1.5, radius = 12, aes(label = seqnames)
  scale_color_manual(values = cols)
.trans <- 0.5
names(.trans) <- "1"
p2 <- ggplot() + layout_circle(gr.crc1, geom = "link", linked.to = "to.gr",
                               aes(color = rearrangements), radius = 7.1, chr.weight = .trans)
          layout_circle(hg19Ideo, geom = "ideo", trackWidth = 1.5,
                color = "gray70", fill = "gray70", chr.weight = .trans) +
          layout_circle(hg19Ideo, geom = "text", trackWidth = 1.5,
                        radius = 12, aes(label = seqnames),
                chr.weight = .trans)+
  scale_color_manual(values = cols)
library(gridExtra)
grid.arrange(p1, p2)
```


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
rearrangements
interchromosomal
intrachromosomal


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
rearrangements
interchromosomal
intrachromosomal

Chapter 7

Manhattan plot

7.1 Introduction

In this tutorial, we introduce a new coordinate system called "genome" for genomic data. This transformation is to put all chromosomes on the same genome coordinates following specified orders and adding buffers in between. One may think about facet ability based on *seqnames*, it can produce something similar to *Manhattan plot*[1], but the view will not be compact. What's more, genome transformation is previous step to form a circular view. In this tutorial, we will simulate some SNP data and use this special coordinate and a specialized function `plotGrandLinear` to make a Manhattan plot.

Manhattan plot is just a special use design with this coordinate system.

7.2 Understand the new coordinate

Let's load some packages and data first

```
library(ggbio)
data(hg19IdeogramCyto, package = "biovizBase")
data(hg19Ideogram, package = "biovizBase")
library(GenomicRanges)
```

Make a minimal example 'GRanges', and see what the default coordiante looks like, pay attention that, by default, the graphics are faceted by 'seqnames' as shown in Figure **??**

[1] http://en.wikipedia.org/wiki/Manhattan

```
library(biovizBase)
gr <- GRanges(rep(c("chr1", "chr2"), each = 5),
              IRanges(start = rep(seq(1, 100, length = 5), times = 2),
                      width = 50))
autoplot(gr, aes(fill = seqnames))
```

What if we specify the coordinate system to be "genome" in `autoplot` function, there is no faceting anymore, the two plots are merged into one single genome space, and properly labeled.

```
autoplot(gr, coord = "genome", aes(fill = seqnames))
```

The internal transformation are implemented into the function `transformToGenome`. And there is some simple way to test if a `GRanges` object is transformed to coordinate "genome" or not

```
gr.t <- transformToGenome(gr)
head(gr.t)
```

```
## GRanges with 6 ranges and 2 metadata columns:
##       seqnames     ranges strand |    .start      .end
##          <Rle>  <IRanges>  <Rle> | <numeric> <numeric>
##   [1]     chr1 [  1,  50]      * |         1        50
##   [2]     chr1 [ 25,  74]      * |        25        74
##   [3]     chr1 [ 50,  99]      * |        50        99
##   [4]     chr1 [ 75, 124]      * |        75       124
##   [5]     chr1 [100, 149]      * |       100       149
##   [6]     chr2 [  1,  50]      * |       180       229
##   ---
##   seqlengths:
##    chr1 chr2
##      NA   NA
```

```
is_coord_genome(gr.t)
```

```
## [1] TRUE
```

```
metadata(gr.t)$coord
```

```
## [1] "genome"
```

7.3 Step 2: Simulate a SNP data set

Let's use the real human genome space to simulate a SNP data set.

```
chrs <- as.character(levels(seqnames(hg19IdeogramCyto)))
seqlths <- seqlengths(hg19Ideogram)[chrs]
set.seed(1)
nchr <- length(chrs)
nsnps <- 100
gr.snp <- GRanges(rep(chrs,each=nsnps),
                  IRanges(start =
                          do.call(c, lapply(chrs, function(chr){
                            N <- seqlths[chr]
                            runif(nsnps,1,N)
                          })), width = 1),
                  SNP=sapply(1:(nchr*nsnps), function(x) paste("rs",x,sep='')),
                  pvalue =  -log10(runif(nchr*nsnps)),
                  group = sample(c("Normal", "Tumor"), size = nchr*nsnps,
                    replace = TRUE)
                  )
genome(gr.snp) <- "hg19"
gr.snp
```

```
## GRanges with 2400 ranges and 3 metadata columns:
##          seqnames                ranges strand   |         SNP
##             <Rle>             <IRanges>  <Rle>   | <character>
##      [1]     chr1 [ 66178199,  66178199]      *   |         rs1
##      [2]     chr1 [ 92752113,  92752113]      *   |         rs2
##      [3]     chr1 [142784056, 142784056]      *   |         rs3
##      [4]     chr1 [226371355, 226371355]      *   |         rs4
##      [5]     chr1 [ 50269347,  50269347]      *   |         rs5
##      ...      ...                   ...    ... ...         ...
##   [2396]     chrY   [34038689, 34038689]      *   |      rs2396
##   [2397]     chrY   [ 3010837,  3010837]      *   |      rs2397
##   [2398]     chrY   [23806602, 23806602]      *   |      rs2398
##   [2399]     chrY   [15474595, 15474595]      *   |      rs2399
##   [2400]     chrY   [10016302, 10016302]      *   |      rs2400
##             pvalue       group
##          <numeric> <character>
##      [1]   1.22380      Normal
##      [2]   1.27916      Normal
##      [3]   0.01199       Tumor
##      [4]   0.09985      Normal
##      [5]   1.49938       Tumor
```

```
##       ...       ...          ...
##   [2396]   0.17601      Normal
##   [2397]   0.78685       Tumor
##   [2398]   0.48952      Normal
##   [2399]   0.60000      Normal
##   [2400]   0.03967      Normal
##   ---
##   seqlengths:
##     chr1 chr10 chr11 chr12 chr13 chr14 ...  chr7  chr8  chr9  chrX  chrY
##       NA    NA    NA    NA    NA    NA ...    NA    NA    NA    NA    NA
```

We use the some trick to make a shorter names.

```
seqlengths(gr.snp)
```

```
##  chr1 chr10 chr11 chr12 chr13 chr14 chr15 chr16 chr17 chr18 chr19  chr2
##    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## chr20 chr21 chr22  chr3  chr4  chr5  chr6  chr7  chr8  chr9  chrX  chrY
##    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
```

```
nms <- seqnames(seqinfo(gr.snp))
nms.new <- gsub("chr", "", nms)
names(nms.new) <- nms
gr.snp <- renameSeqlevels(gr.snp, nms.new)
seqlengths(gr.snp)
```

```
##  1 10 11 12 13 14 15 16 17 18 19  2 20 21 22  3  4  5  6  7  8  9  X  Y
## NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
```

7.4 Step 3: Start to make Manhattan plot by using `autoplot`

wrapped basic functions into **`autoplot`**, you can specify the coordinate. Figure **??** shows what the unordered object looks like.

```
autoplot(gr.snp, coord = "genome", geom = "point", aes(y = pvalue), space.skip = 0.01)
```

That's probably not what you want, if you want to change to specific order, just sort them by hand and use 'keepSeqlevels'. Figure **??** shows a sorted plot.

```
gr.snp <- keepSeqlevels(gr.snp, c(1:22, "X", "Y"))
values(gr.snp)$highlight <- FALSE
idx <- sample(1:length(gr.snp), size = 15)
values(gr.snp)$highlight[idx] <- TRUE
values(gr.snp)$id <- 1:length(gr.snp)
p <- autoplot(gr.snp, coord = "genome", geom = "point", aes(y = pvalue), space.skip = 0.01)
```

NOTICE: the data now doesn't have information about lengths of each chromosomes, this is allowed to be plotted, but it's misleading sometimes, without chromosomes lengths information, *ggbio* use data space to make estimated lengths for you, this is not accurate! So let's just assign `seqlengths` to the object. Then you will find the data space now is distributed proportional to real space as shown in Figure **??**.

```
names(seqlths) <- gsub("chr", "", names(seqlths))
seqlengths(gr.snp) <- seqlths[names(seqlengths(gr.snp))]
## backup
gr.back <- gr.snp
autoplot(gr.snp, coord = "genome", geom = "point", aes(y = pvalue), space.skip = 0.01)
```

In `autoplot`, argument `coord` is just used to transform the data, after that, you can use it as common `GRanges`, all other geom/stat works for it. Here just show a simple example for another geom "line" as shown in Figure **??**

```
autoplot(gr.snp, coord = "genome", geom = "line", aes(y = pvalue, group = seqnames,
                                        color = seqnames))
```

7.5 Convenient `plotGrandLinear` function

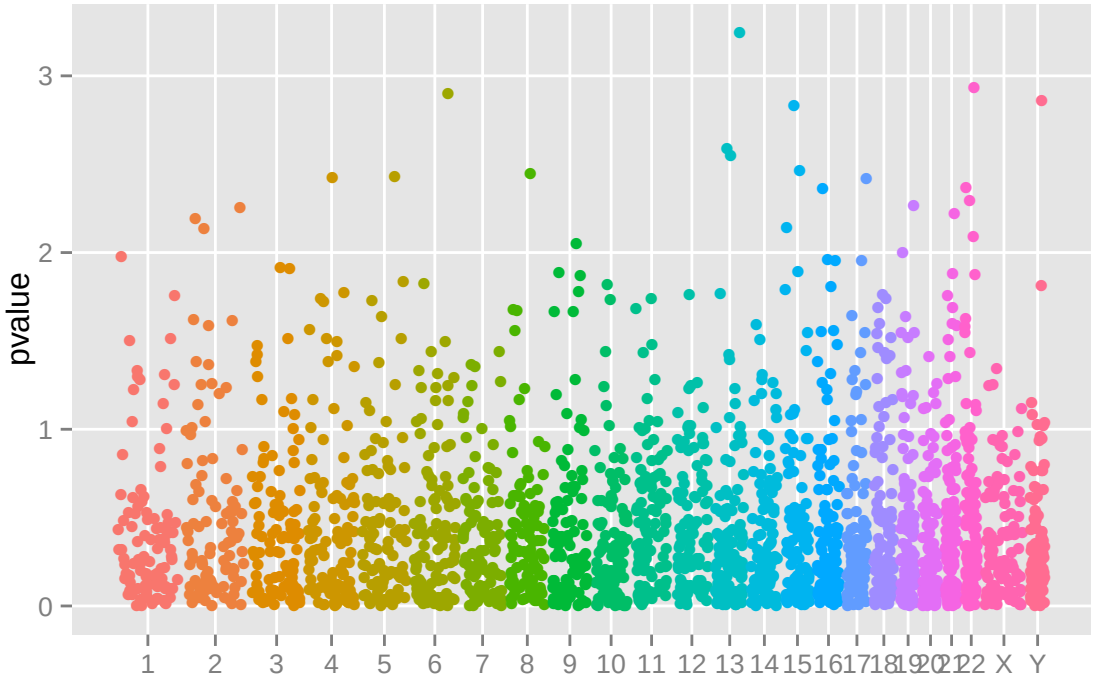
In *ggbio*, sometimes we develop specialized function for certain types of plots, it's basically a wrapper over lower level API and `autoplot`, but more convenient to use. Here for *Manhattan plot*, we have a function called `plotGrandLinear` used for it. aes(y =) is required to indicate the y value, e.g. p-value. Figure **??** shows a defalut graphic.

Color mapping is automatically figured out by *ggbio* following the rules

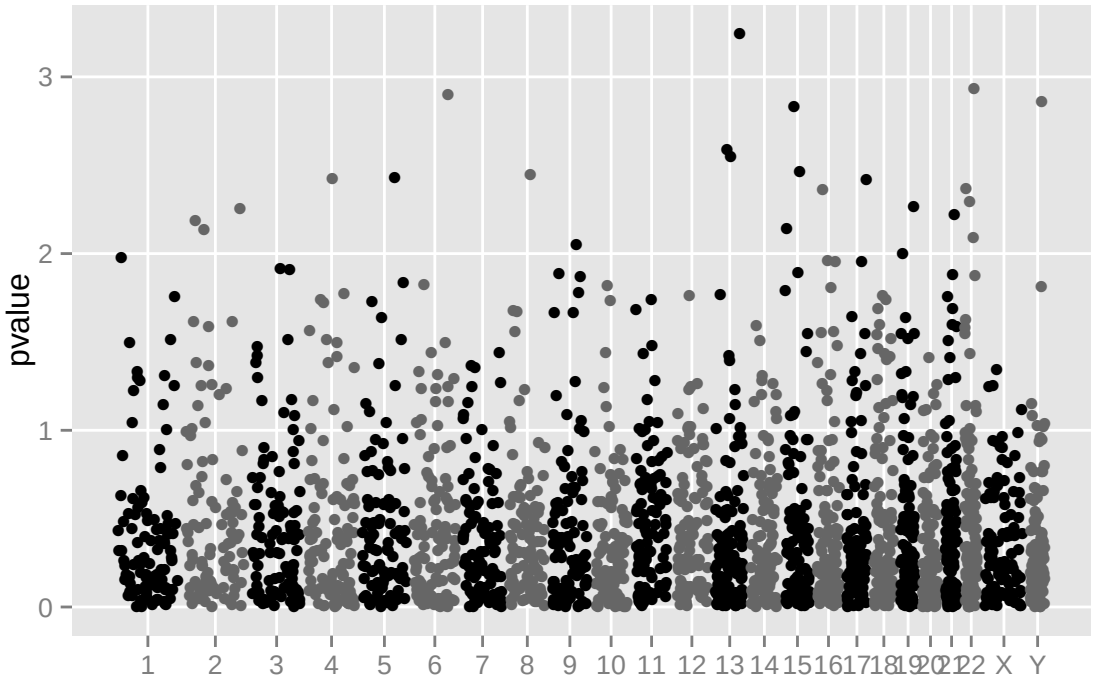
- if `color` present in `aes()`, like `aes(color = seqnames)`, it will assume it's mapping to data column called 'seqnames'.
- if `color` is not wrapped in `aes()`, then this function will **recylcle** them to all chromosomes.
- if `color` is single character representing color, then just use one arbitrary color.

Let's test some examples for controling colors.

```
plotGrandLinear(gr.snp, aes(y = pvalue, color = seqnames))

## Object of class "ggbio"
```

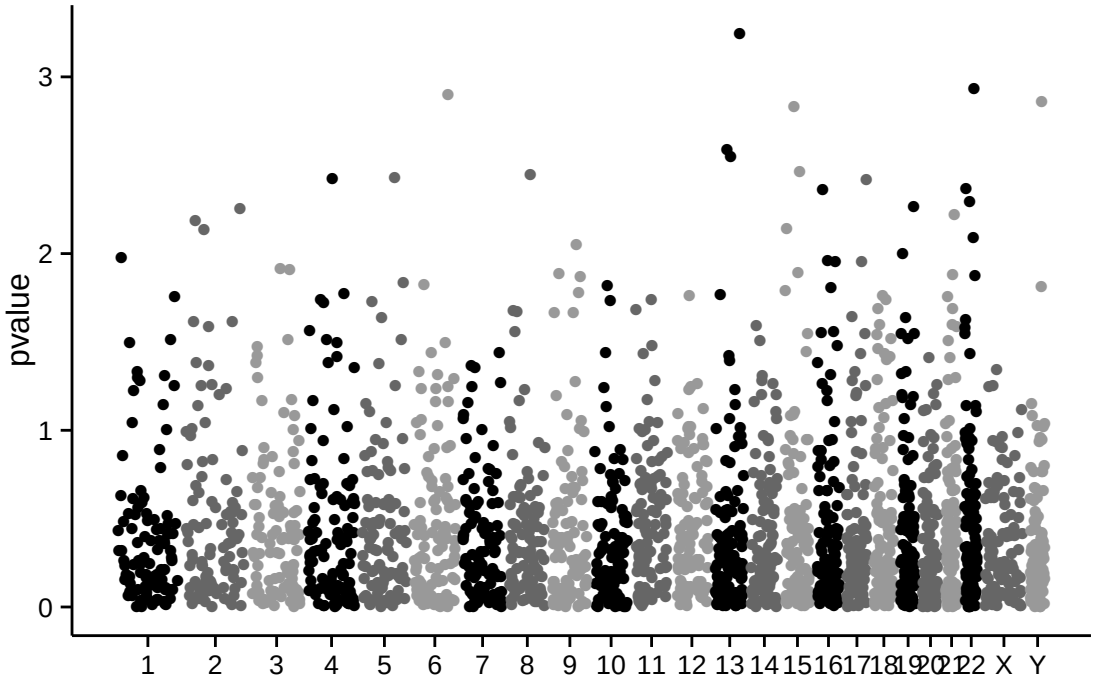



```
plotGrandLinear(gr.snp, aes(y = pvalue), color = c("gray0", "gray40"))

## Object of class "ggbio"
```



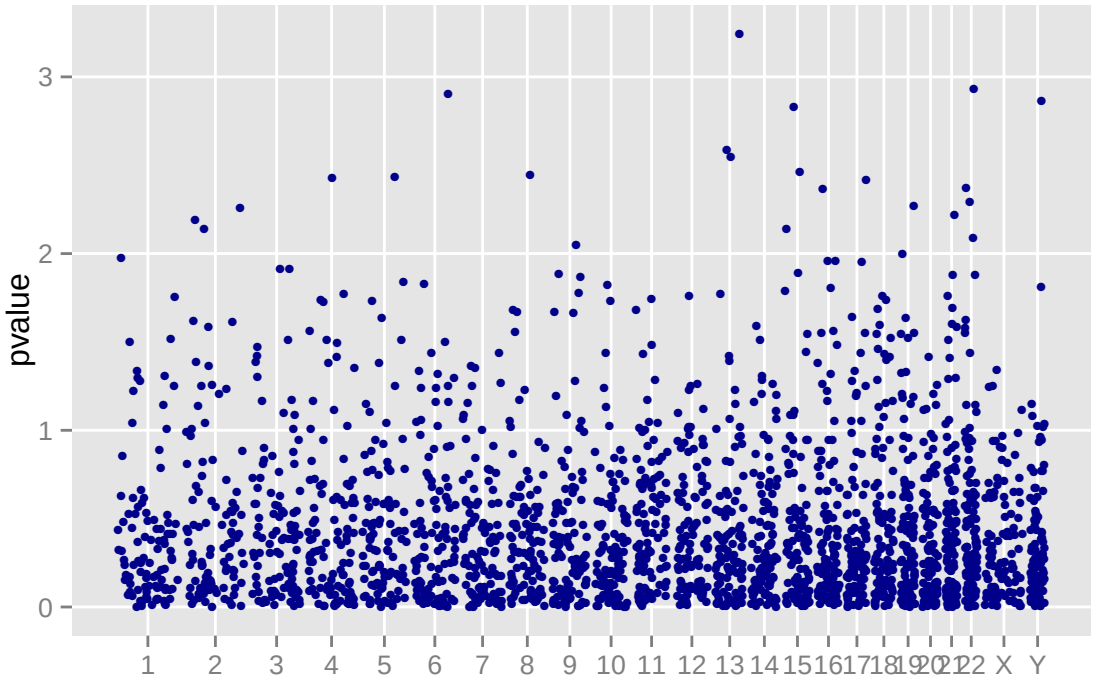
Even more than two colors.

```
plotGrandLinear(gr.snp, aes(y = pvalue), color = c("gray0", "gray40", "gray60")) +
  theme_classic() + theme(legend.position = "none")


## Object of class "ggbio"
```



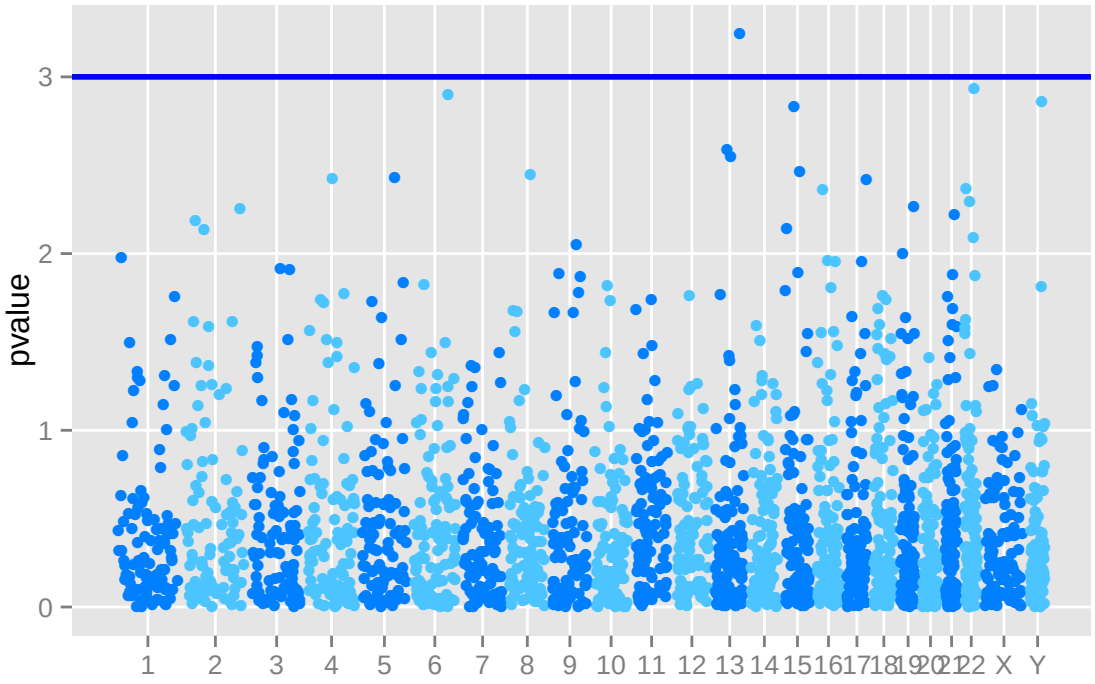
For fixed color, and smaller point

```
plotGrandLinear(gr.snp, aes(y = pvalue), color = "darkblue", size = 1.5)

## Object of class "ggbio"
```



You can also add cutoff line as shown in Figure **??**.

```
plotGrandLinear(gr.snp, aes(y = pvalue), cutoff = 3, cutoff.color = "blue", cutoff.size = 1)

## Object of class "ggbio"
```



This is equivalent to *ggplot2* 's API.

```
plotGrandLinear(gr.snp, aes(y = pvalue)) + geom_hline(yintercept = 3, color = "blue", size =
```

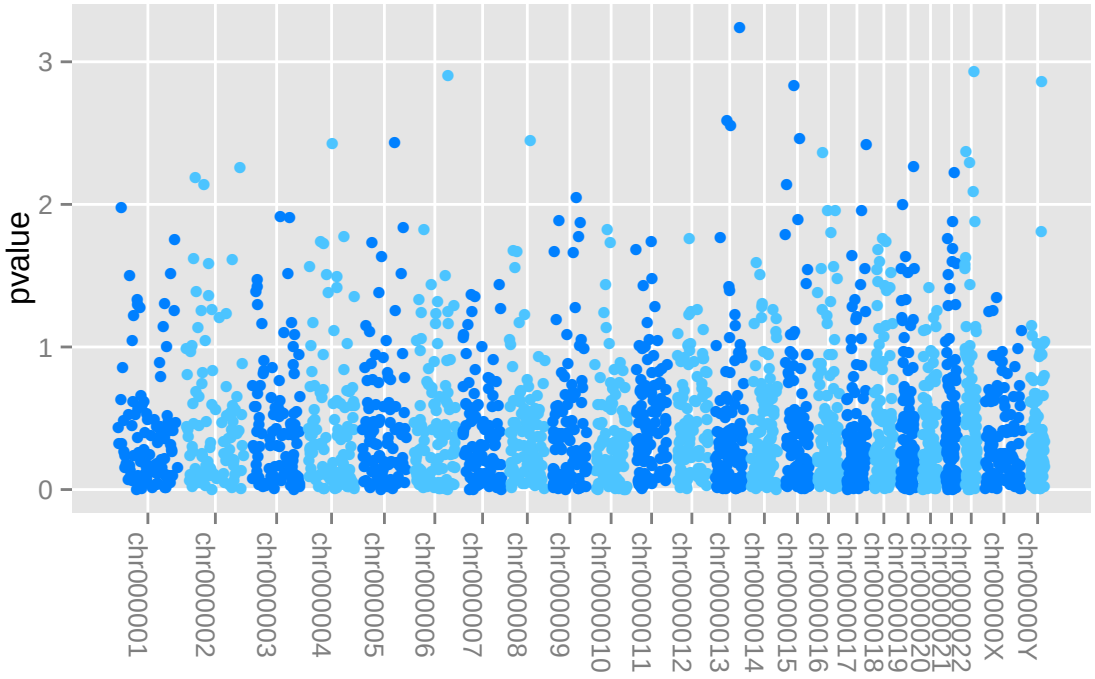
Sometimes the names of chromosomes maybe very long, you may want to rotate them, let's make a longer name first

```
## let's make a long name
nms <- seqnames(seqinfo(gr.snp))
nms.new <- paste("chr00000", nms, sep = "")
names(nms.new) <- nms
gr.snp <- renameSeqlevels(gr.snp, nms.new)
seqlengths(gr.snp)
```

```
##  chr000001  chr000002  chr000003  chr000004  chr000005  chr000006
##  249250621  243199373  198022430  191154276  180915260  171115067
##  chr000007  chr000008  chr000009 chr0000010 chr0000011 chr0000012
##  159138663  146364022  141213431  135534747  135006516  133851895
## chr0000013 chr0000014 chr0000015 chr0000016 chr0000017 chr0000018
##  115169878  107349540  102531392   90354753   81195210   78077248
## chr0000019 chr0000020 chr0000021 chr0000022  chr00000X  chr00000Y
##   59128983   63025520   48129895   51304566  155270560   59373566
```

Then rotate it!

```
plotGrandLinear(gr.snp, aes(y = pvalue)) + theme(axis.text.x=theme_text(angle=-90, hjust=0))

## Object of class "ggbio"
```

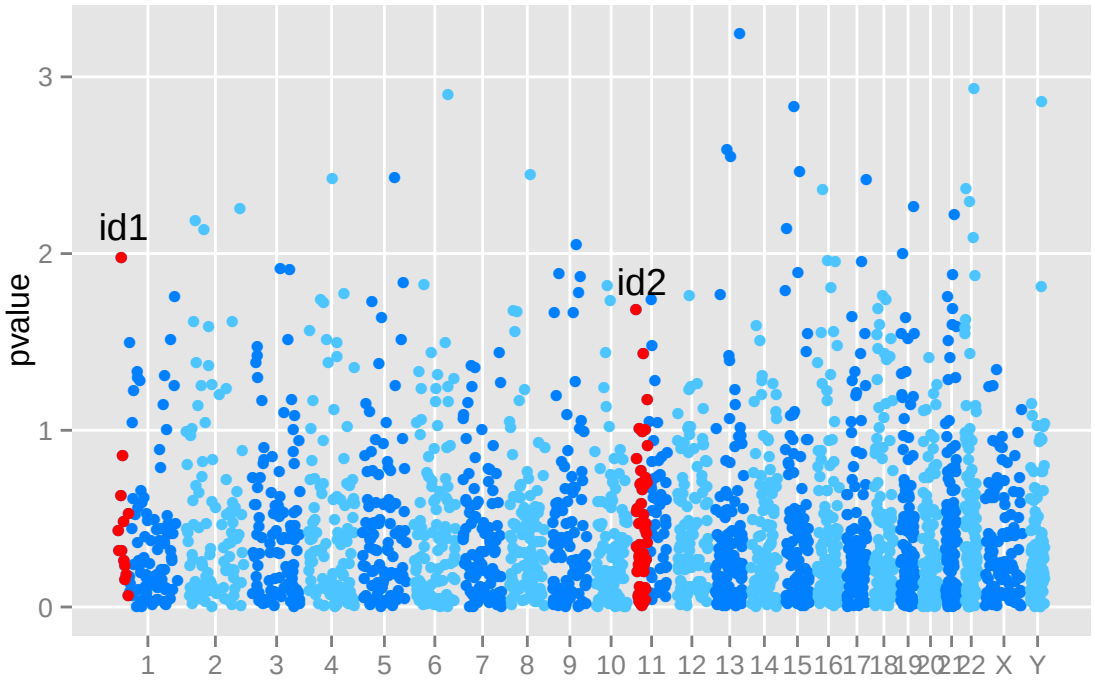


As you can tell from above examples, all utilities works for *ggplot2* will work for *ggbio* too.

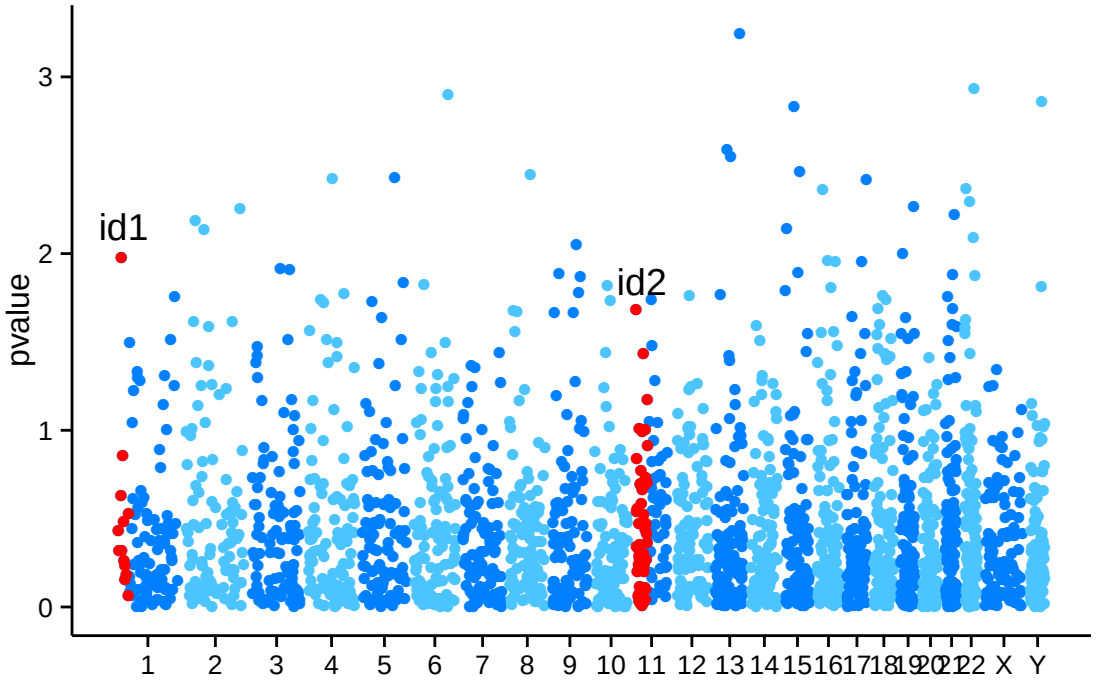
7.6 Annotating manhattan plot easily

You can provide a highlight *GRanges*, and each row highlights a set of overlaped snps, and labeled by rownames or certain columns, there is more control in the function as parameters, with prefix highlight.*, so you could control color, label size and color, etc.

```
gr.snp <- gr.back
gro <- GRanges(c("1", "11"), IRanges(c(100, 2e6), width = 5e7))
names(gro) <- c("id1", "id2")
plotGrandLinear(gr.snp, aes(y = pvalue), highlight.gr = gro)

## Object of class "ggbio"
```




```
plotGrandLinear(gr.snp, aes(y = pvalue), highlight.gr = gro) + theme_classic() +
  theme(legend.position  = "none")


## Object of class "ggbio"
```

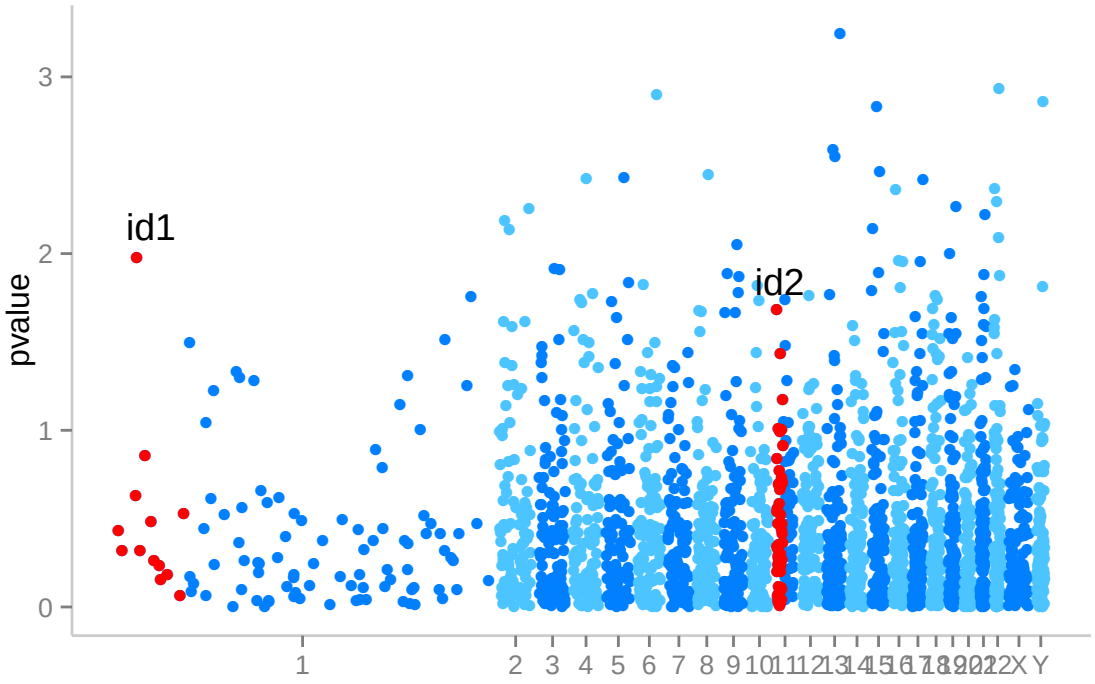


7.7 Unequal space

This is an experimental feature that added after 1.7.12, which transform the genome space to some specified proportions.

In `plotGrandLinear`, there is a new parameter called `chr.weight`, which is a vector of numeric value and sum of those value should not exceed 1, these value indicates proportion of chrommosome space to take in overall space. Names of this vectors are chromosomes names, and you can only specify a few of them, other chromosomes will take up left space according to their space.

```
.trans <- 0.5
names(.trans) <- "1"
plotGrandLinear(gr.snp, aes(y = pvalue), highlight.gr = gro, chr.weight = .trans) +
  theme_clear() +  theme(legend.position  = "none")


## Object of class "ggbio"
```


pvalue
3
2
1
0
id1
id2
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
X
Y

Chapter 8

Karyogram overview

8.1 Introduction

A karyotype is the number and appearance of chromosomes in the nucleus of a eukaryotic cell[1]. It's one kind of overview when we want to show distribution of certain events on the genome, for example, binding sites for certain protein, even compare them acroos samples as example shows in this section.

`GRanges` and `Seqinfo` object are also an ideal container for storing data needed for karyogram plot. Here is the strategy we used for generating ideogram templates.

- Althouth `seqlengths` is not required, it's highly recommended for plotting karyogram. If a `GRanges` object contains `seqlengths`, we know exactly how long each chromosome is, and will use this information to plot genome space, particularly we plot all levels included in it, **NOT JUST** data space.
- If a `GRanges` has no `seqlengths`, we will issue a warning and try to estimate the chromosome lengths from data included. This is **NOT** accurate most time, so please pay attention to what you are going to visualize and make sure set `seqlengths` before hand.

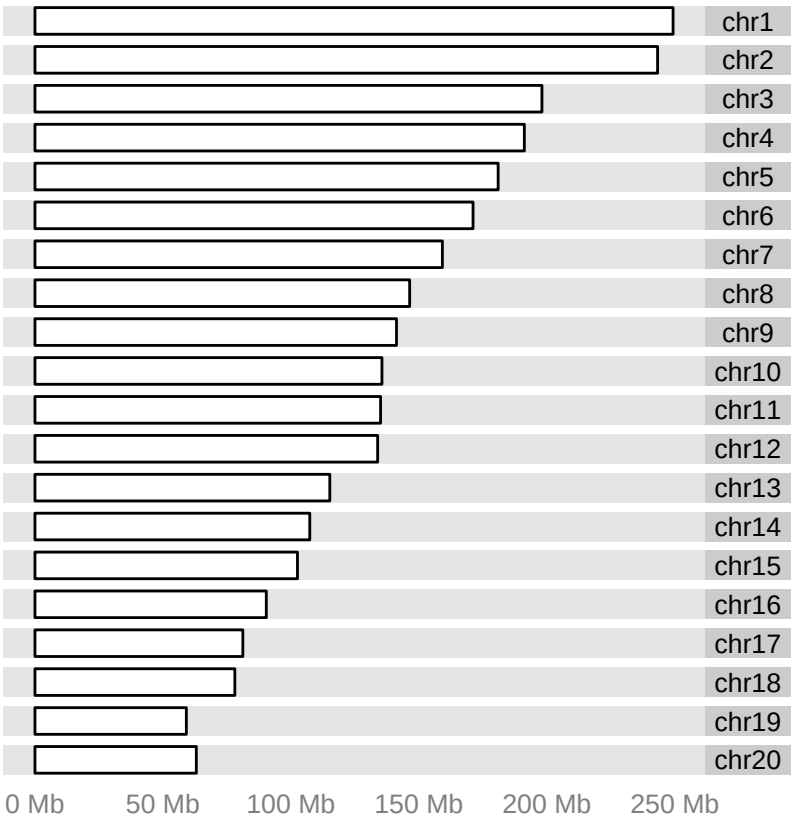
8.2 `autoplot`

Let's first introduce how to use `autoplot` to generate karyogram graphic.

The most easy one is to just plot Seqinfo by using `autoplot`, if your *GRanges* object has seqinfo with seqlengths information.

[1] http://en.wikipedia.org/wiki/Karyotype

```
data(hg19Ideogram, package = "biovizBase")
chrs <- paste0("chr", 1:20)
p <- autoplot(seqinfo(hg19Ideogram)[chrs])
p
```



Even more typical karyogram overview with cytoband, this will even show the arms, two required columns are required 'name' and 'gieStain'.

```
data(hg19IdeogramCyto, package = "biovizBase")
head(hg19IdeogramCyto)


## GRanges with 6 ranges and 2 metadata columns:
##       seqnames               ranges strand |     name gieStain
##          <Rle>            <IRanges>  <Rle> | <factor> <factor>
##   [1]     chr1 [       0,  2300000]      * |   p36.33     gneg
##   [2]     chr1 [ 2300000,  5400000]      * |   p36.32   gpos25
##   [3]     chr1 [ 5400000,  7200000]      * |   p36.31     gneg
##   [4]     chr1 [ 7200000,  9200000]      * |   p36.23   gpos25
##   [5]     chr1 [ 9200000, 12700000]      * |   p36.22     gneg
```

```
##   [6]     chr1 [12700000, 16200000]      * |   p36.21   gpos50
##   ---
##   seqlengths:
##     chr1 chr10 chr11 chr12 chr13 chr14 ...  chr7  chr8  chr9  chrX  chrY
##       NA    NA    NA    NA    NA    NA ...    NA    NA    NA    NA    NA
```

```
p <- autoplot(hg19IdeogramCyto, layout = "karyogram", cytoband = TRUE)
```

Tips: Your turn: change the order of chromosomes.

We use a default data in package *biovizBase*, which is a subset of RNA editing set in human. The data involved in this `GRanges` is sparse, so we cannot simply use it to make karyogram, otherwise, the estimated chromosome lengths will be very rough and inaccurate. So what we need to do is:

1. Adding seqlegnths to this `GRanges` object. If you adding seqlengths to object, we have two ways to show chromosome space as karyogram.
   `autoplot(object, layout = 'karyogram')` or
   `autoplot(seqinfo(object))`.
2. Changing the order of chromosomes.
3. Visualize it and map variable to different aesthetics.

```
data(darned_hg19_subset500, package = "biovizBase")
dn <- darned_hg19_subset500
head(dn)
```

```
## GRanges with 6 ranges and 10 metadata columns:
##       seqnames                ranges strand |       inchr       inrna
##          <Rle>             <IRanges>  <Rle> | <character> <character>
##   [1]     chr5 [ 86618225,  86618225]      - |           A           I
##   [2]     chr7 [ 99792382,  99792382]      - |           A           I
##   [3]    chr12 [110929076, 110929076]      - |           A           I
##   [4]    chr20 [ 25818128,  25818128]      - |           A           I
##   [5]     chr3 [132397992, 132397992]      + |           A           I
##   [6]    chr19 [ 59078471,  59078471]      - |           A           I
##               snp        gene      seqReg       exReg
##       <character> <character> <character> <character>
##   [1]        <NA>        <NA>           0        <NA>
##   [2]        <NA>        <NA>           0        <NA>
```

```
##   [3]        <NA>        <NA>           0        <NA>
##   [4]        <NA>        <NA>           0        <NA>
##   [5]        <NA>        <NA>           0        <NA>
##   [6]        <NA>        MZF1           I        <NA>
##                  source      ests      esta      author
##              <character> <integer> <integer> <character>
##   [1]           amygdala         0         0    15342557
##   [2]               <NA>         0         0    15342557
##   [3]     salivary gland         0         0    15342557
##   [4] brain, hippocampus         0         0    15342557
##   [5]    small intestine         0         0    15342557
##   [6]               <NA>         0         0    15258596
##   ---
##   seqlengths:
##     chr1 chr10 chr11 chr12 chr13 chr14 ...  chr6  chr7  chr8  chr9  chrX
##       NA    NA    NA    NA    NA    NA ...    NA    NA    NA    NA    NA
```

```
## add seqlengths
## we have seqlegnths information in another data set
data(hg19Ideogram, package = "biovizBase")
seqlengths(dn) <- seqlengths(hg19Ideogram)[names(seqlengths(dn))]
## now we have seqlengths
head(dn)
```

```
## GRanges with 6 ranges and 10 metadata columns:
##       seqnames                ranges strand |       inchr       inrna
##          <Rle>             <IRanges>  <Rle> | <character> <character>
##   [1]     chr5 [ 86618225,  86618225]      - |           A           I
##   [2]     chr7 [ 99792382,  99792382]      - |           A           I
##   [3]    chr12 [110929076, 110929076]      - |           A           I
##   [4]    chr20 [ 25818128,  25818128]      - |           A           I
##   [5]     chr3 [132397992, 132397992]      + |           A           I
##   [6]    chr19 [ 59078471,  59078471]      - |           A           I
##                snp        gene      seqReg       exReg
##       <character> <character> <character> <character>
##   [1]        <NA>        <NA>           0        <NA>
##   [2]        <NA>        <NA>           0        <NA>
##   [3]        <NA>        <NA>           0        <NA>
##   [4]        <NA>        <NA>           0        <NA>
##   [5]        <NA>        <NA>           0        <NA>
##   [6]        <NA>        MZF1           I        <NA>
##                  source      ests      esta      author
##              <character> <integer> <integer> <character>
##   [1]           amygdala         0         0    15342557
```

```
##   [2]               <NA>         0         0    15342557
##   [3]     salivary gland         0         0    15342557
##   [4] brain, hippocampus         0         0    15342557
##   [5]    small intestine         0         0    15342557
##   [6]               <NA>         0         0    15258596
##   ---
##   seqlengths:
##         chr1     chr10     chr11 ...      chr8      chr9      chrX
##    249250621 135534747 135006516 ... 146364022 141213431 155270560


## then we change order
dn <- keepSeqlevels(dn, paste0("chr", c(1:22, "X")))
autoplot(dn, layout = "karyogram")
## this equivalent to
## autoplot(seqinfo(dn))
```

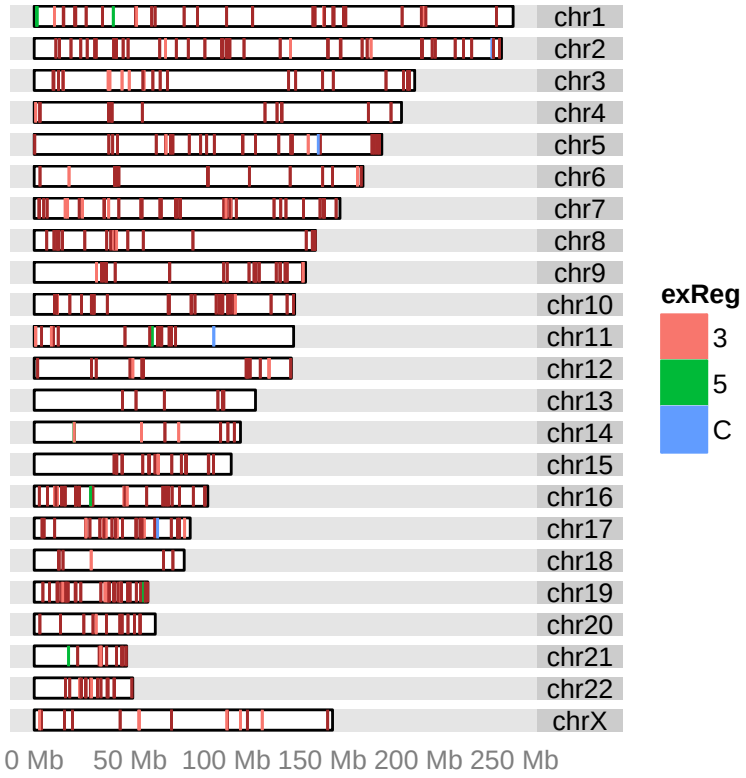
Then we take one step further, the power of *ggplot2* or *ggbio* is the flexible multivariate data mapping ability in graphics, make data exploration much more convenient. In the following example, we are trying to map a categorical variable 'exReg' to color, this variable is included in the data, and have three levels, '3' indicate 3' utr, '5' means 5' utr and 'C' means coding region. We have some missing values indicated as `NA`, in default, it's going to be shown in gray color, and keep in mind, since the basic geom(geometric object) is rectangle, and genome space is very large, so change both color/fill color of the rectangle to specify both border and filled color is necessary to get the data shown as different color, otherwise if the region is too small, border color is going to override the fill color.

```
## since default is geom rectangle, even though it's looks like segment
## we still use both fill/color to map colors
autoplot(dn, layout = "karyogram", aes(color = exReg, fill = exReg))
```

Or you can set the missing value to particular color you want.

Note: NA values is not shown on the legend.

```
## since default is geom rectangle, even though it's looks like segment
## we still use both fill/color to map colors
autoplot(dn, layout = "karyogram", aes(color = exReg, fill = exReg)) +
  scale_color_discrete(na.value = "brown")


## Object of class "ggbio"
```



8.3 plotKaryogram

`plotKaryogram` (or `plotStackedOverview`) are specialized function to draw karyogram graphics. It's actually what function `autoplot` calls inside. API is a littler simpler because layout 'karyogram' is default in these two functions. So equivalent usage is like

```
plotKaryogram(dn)
plotKaryogram(dn, aes(color = exReg, fill = exReg))
```

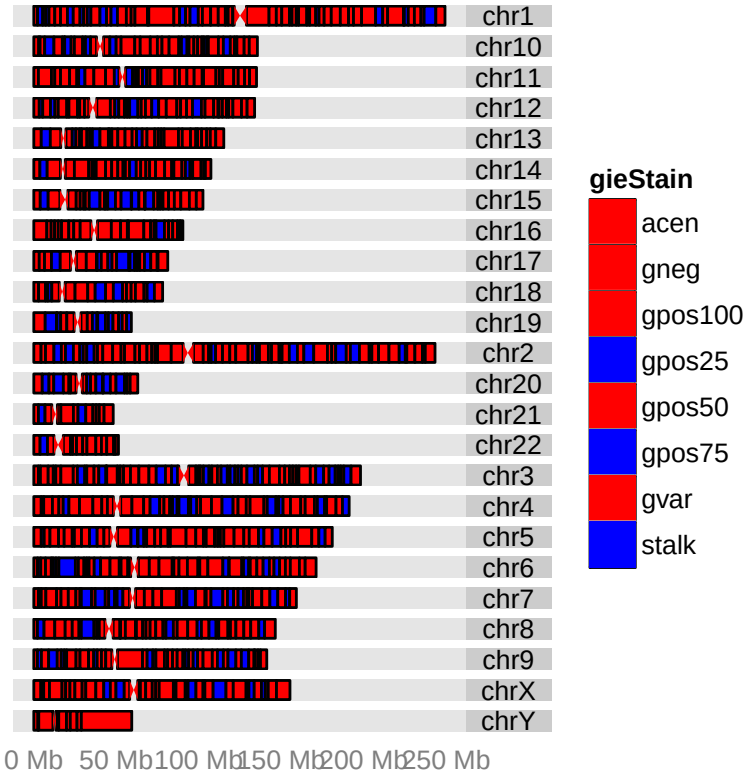
8.4 layout_karyogram

In this section, a lower level function `layout_karyogram` is going to be introduced. This is convenient API for constructing karyogram plot and adding more data layer by layer. Function `ggplot` is just to create blank object to add layer on.

You need to pay attention to

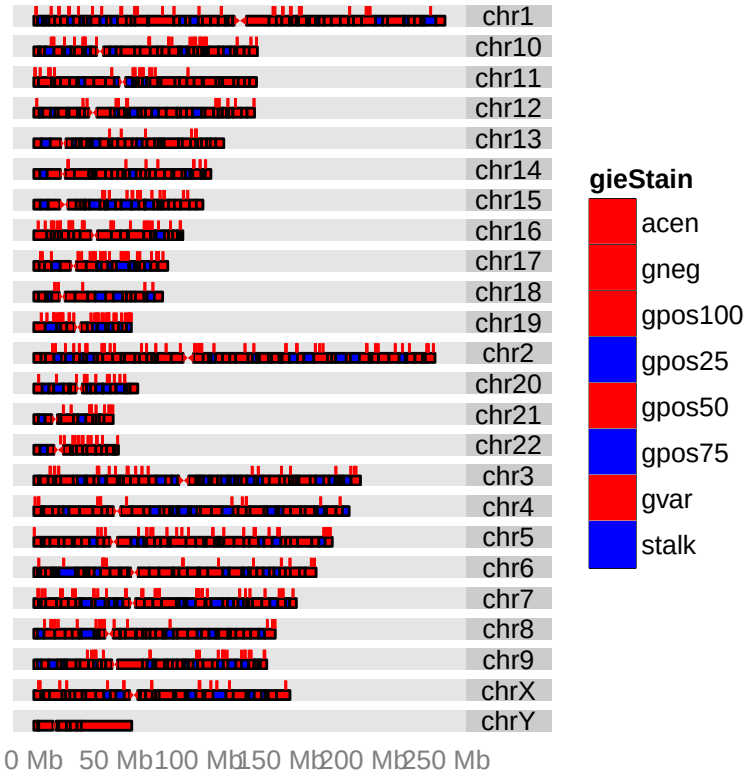
- when you add plots layer by layer, seqnames of different data must be the same to make sure the data are mapped to the same chromosome. For example, if you name chromosome following schema like *chr1* and use just number *1* to name other data, they will be treated as different chromosomes.
- cannot use the same aesthetics mapping multiple time for different data. For example, if you have used aes(color =), for one data, you cannot use aes(color =) anymore for mapping variables from other add-on data, this is currently not allowed in *ggplot2*, even though you expect multiple color legend shows up, this is going to confuse people which is which. HOWEVER, `color` or `fill` without `aes()` wrap around, is allowed for any track, it's set single arbitrary color. This is shown in Figure **??**.
- Default rectangle y range is [0, 10], so when you add on more data layer by layer on existing graphics, you can use `ylim` to control how to normalize your data and plot it relative to chromosome space. For example, with default, chromosome space is plotted between y [0, 10], if you use `ylim = c(10 , 20)`, you will stack data right above each chromosomes and with equal width. For geom like 'point', which you need to specify 'y' value in `aes()`, we will add 5% margin on top and at bottom of that track.

```
## plot ideogram
p <- ggplot(hg19) + layout_karyogram(cytoband = TRUE)

## Error:  error in evaluating the argument 'data' in selecting a method for function
'ggplot':  Error:  object 'hg19' not found

p

## Object of class "ggbio"
```



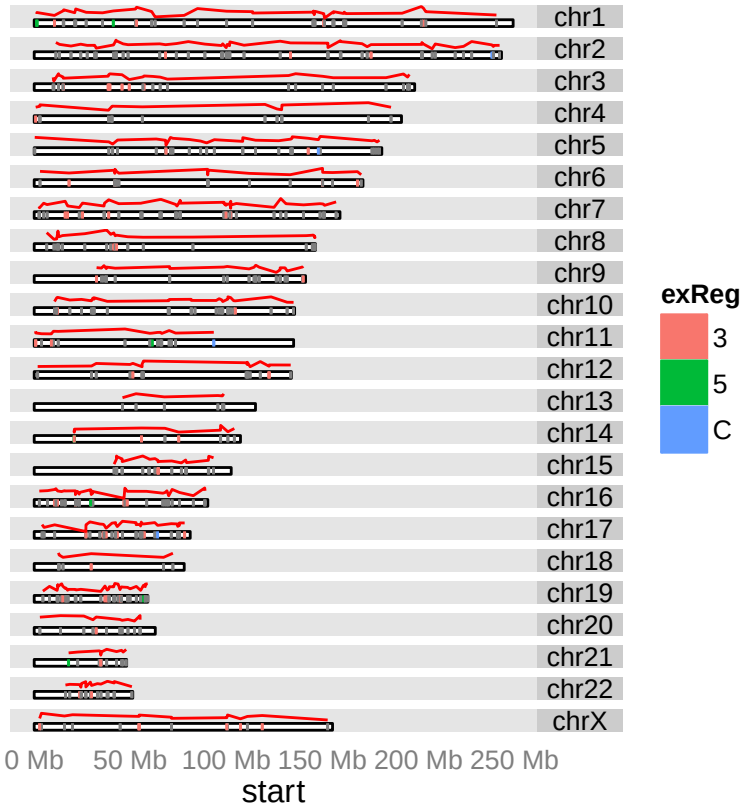

```
## eqevelant autoplot(hg19, layout = "karyogram", cytoband = TRUE)
```

```
p <- p + layout_karyogram(dn, geom = "rect", ylim = c(11, 21), color = "red")
## commented line below won't work
## the cytoband fill color has been used already.
## p <- p + layout_karyogram(dn, aes(fill = exReg, color = exReg), geom = "rect")
p


## Object of class "ggbio"
```

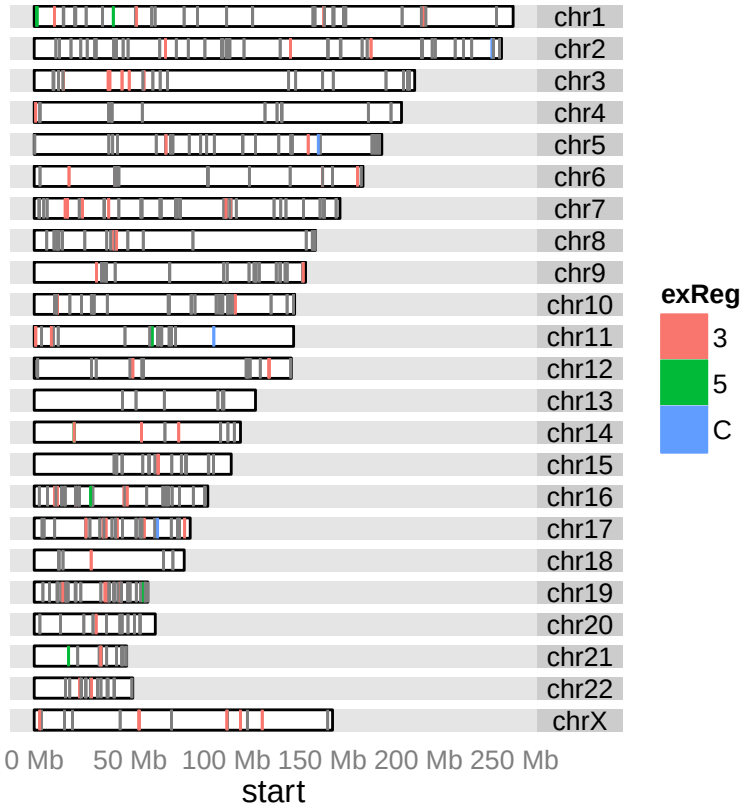


Then we construct another multiple layer graphics for multiple data using different geom, suppose we want to show RNA-editing sites on chromosome space as rectangle(looks like segment in graphic) and stack a line for another track above.

```
## plot chromosome space
p <- autoplot(seqinfo(dn))
## make sure you pass rect as geom
## otherwise you just get background
p <- p + layout_karyogram(dn, aes(fill = exReg, color = exReg), geom = "rect")
values(dn)$pvalue <- rnorm(length(dn))
p + layout_karyogram(dn, aes(x = start, y = pvalue),
                     ylim = c(10, 30), geom = "line", color = "red")
```


chr1
chr2
chr3
chr4
chr5
chr6
chr7
chr8
chr9
chr10
chr11
chr12
chr13
chr14
chr15
chr16
chr17
chr18
chr19
chr20
chr21
chr22
chrX
exReg
3
5
C
0 Mb
50 Mb
100 Mb
150 Mb
200 Mb
250 Mb
start

```
p
```


chr1
chr2
chr3
chr4
chr5
chr6
chr7
chr8
chr9
chr10
chr11
chr12
chr13
chr14
chr15
chr16
chr17
chr18
chr19
chr20
chr21
chr22
chrX
exReg
3
5
C
0 Mb
50 Mb
100 Mb
150 Mb
200 Mb
250 Mb
start

Chapter 9

Visualize genomic features

9.1 Introduction

Transcript-centric annotation is one of the most useful tracks that frequently aligned with other data in many genome browsers. In Bioconductor, you can either request data on the fly from UCSC or BioMart, which require internet connection, or you can save frequently used annotation data of particular organism, for example human genome, as a local data base. Package *GenomicFeatures* provides very convenient API for making and manipulating such database. Bioconductor also pre-built some frequently used genome annotation as packages for easy installation, for instance, for human genome(hg19), there is a meta data package called *TxDb.Hsapiens.UCSC.hg19.knownGene*, after you load this package, a `TranscriptDb` object called `TxDb.Hsapiens.UCSC.hg19.knownGene` will be visible from your workspace. This object contains information like coding regions, exons, introns, utrs, transcripts for this genome. If you cannot find the organism you want in Bioconductor meta packages, please refer to the vignette of package *GenomicFeatures* to check how to build your own data base manually.

ggbio providing visualization utilities based on this specific object, in the following tutorial we cover some usage:

- How to plot genomic features for certain region, including coding region, introns, utrs.
- How to change geom of introns, how to revise arrow size and density.
- How to change aesthetics such as colors.
- How to plot single genomic features by make statistical transformation of "reduce".
- How to revise y label using expression and pattern.
- How to change x-scale unit to arbitrary *kb,bp*.
- How to use lower level API.

9.2 Usage

9.2.1 autoplot

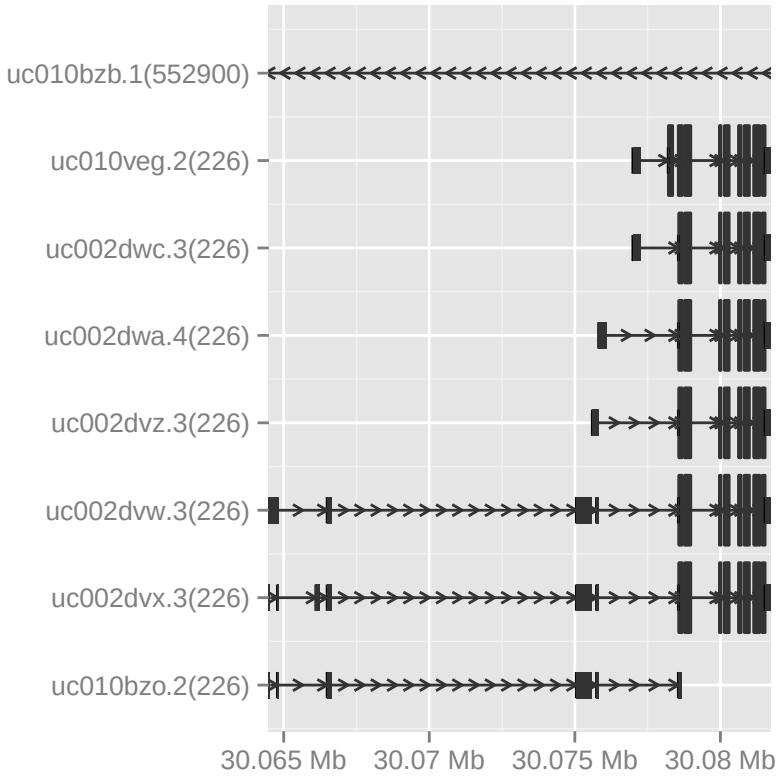
`autoplot` API is higher level API in *ggbio* which tries to make smart decision for object-oriented graphics. Another vignette have more detailed introduction to this function.

In this tutorial, we solely focus on visualization of `TranscriptDb` object.

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
## suppose you already know the region you want to visualize
## or for human genome, you can try following commented code
## data(genesymbol, package = "biovizBase")
## genesymbol["ALDOA"]
aldoa.gr <- GRanges("chr16", IRanges(30064491, 30081734))
aldoa.gr
```

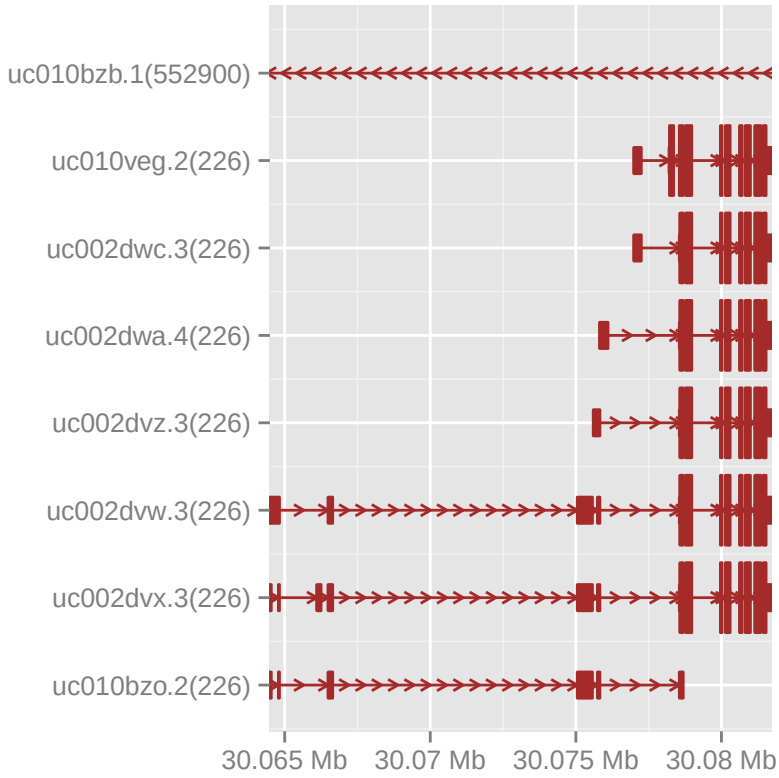
```
## GRanges with 1 range and 0 metadata columns:
##       seqnames              ranges strand
##          <Rle>           <IRanges>  <Rle>
##   [1]    chr16 [30064491, 30081734]      *
##   ---
##   seqlengths:
##    chr16
##       NA
```

```
library(ggbio)
p1 <- autoplot(txdb, which = aldoa.gr)
p1
```



You can changing some aesthetics like colors in `autoplot`, since rectangle is defined by 'color' which is border color and 'fill' for filled color.

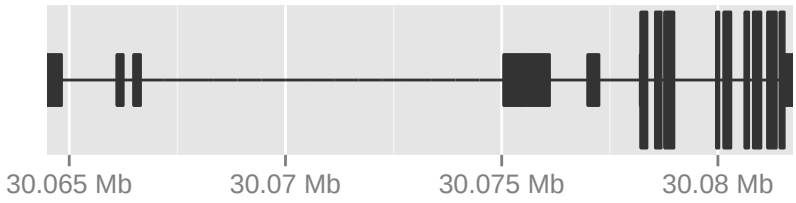
```
library(ggbio)
p1 <- autoplot(txdb, which = aldoa.gr, fill = "brown", color = "brown")
p1
```



`autoplot` function for object `TranscriptDb` has two supported statistical transformation.

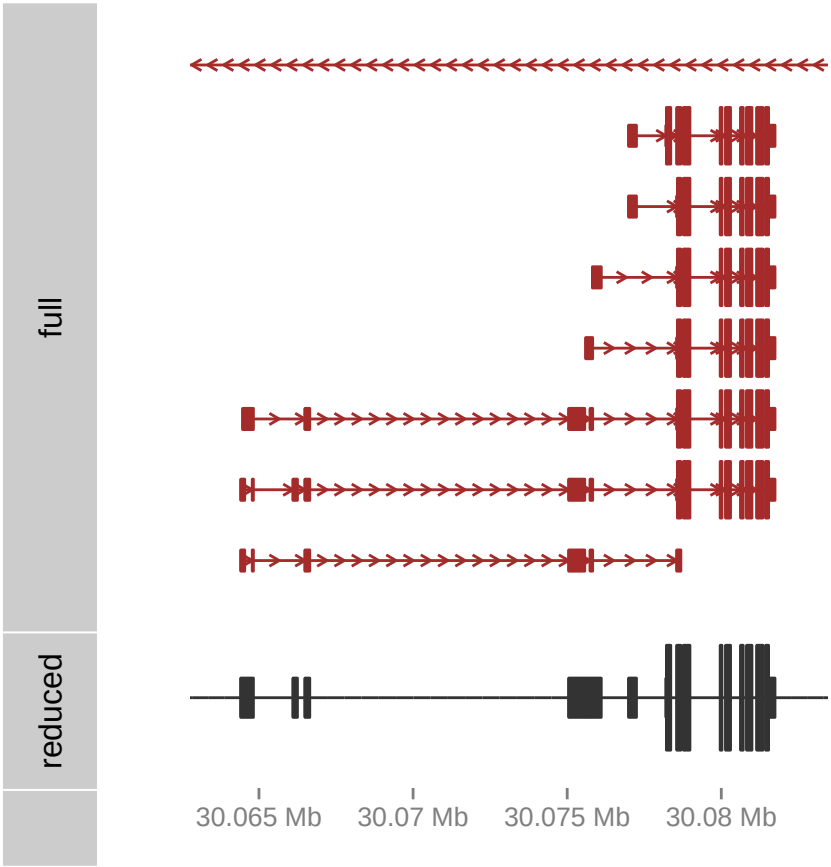
- **identity**: full model, show each transcript, parsing coding region, introns and utrs automatically from the database. intorns are shown as small arrows to indicate the direction, exons are represented as wider rectangles and utrs are represented as narrow rectangles. This transformation is shown in Figure **??**
- **reduce**: reduced model, show single reduced model, which take union of CDS, utrs and re-compute introns, as shown in Figure **??**.

```
p2 <- autoplot(txdb, which = aldoa.gr, stat = "reduce")
print(p2)
```



To better understand the behavior of "reduce" transformation, we layout these two graphics by tracks as shown in Figure **??**. Function `Tracks` has been introduced in detail in another vignette.

```
tracks(full = p1, reduced = p2, heights = c(4,1)) +
  theme_alignment(grid=FALSE, border = FALSE)
```



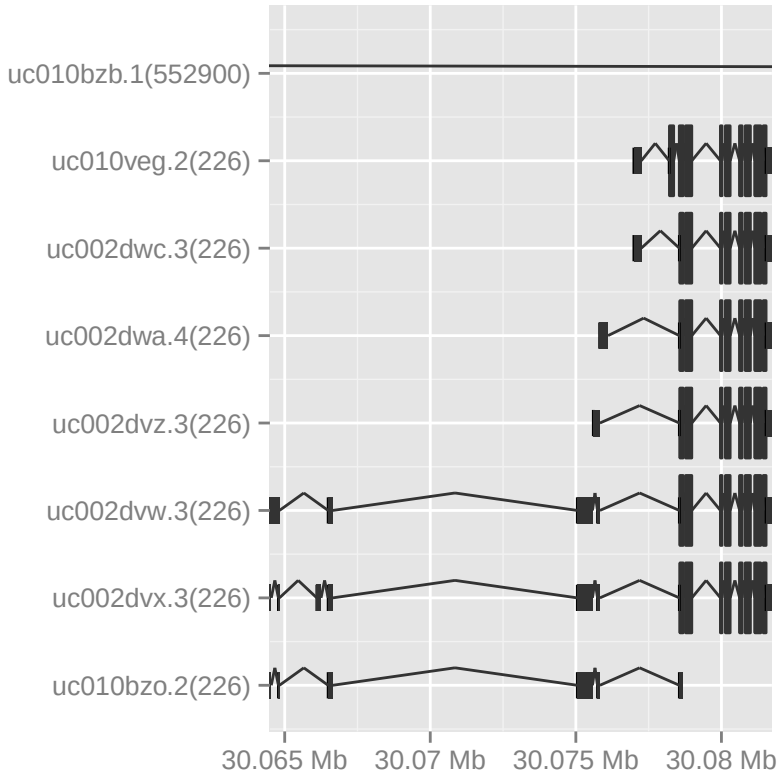
We allow users to change the way to visualization introns here, it's controlled by parameter "gap.geom", supported three geoms:

- **arrow**: with small arrow to indicate the strand direction, extra parameter existing to control the appearance of the arrow, as shown in Figure **??**. **arrow.rate** control how dense the arrows shows in between.

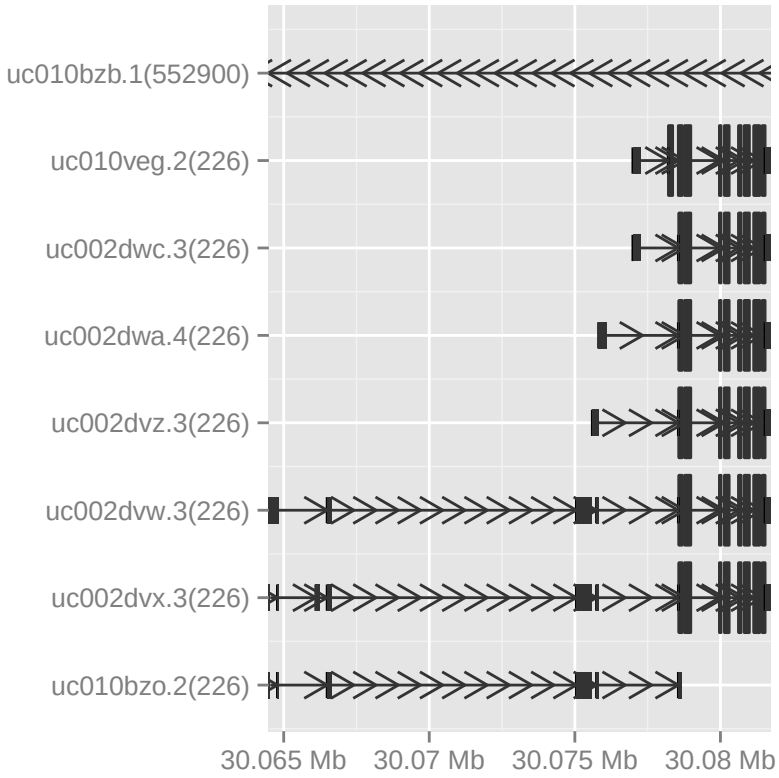
- **chevron**:chevron to show as introns, no strand indication. please check `geom_chevron`.
- **segment**:segments to show as introns, no strand indication.

The geometric object for ranges, introns and uts are controled by parameters `range.geom, gap.geom, utr.geom`. For example if you want to change the geom for gap, just change the `gap.geom`.

```
autoplot(txdb, which = aldoa.gr, gap.geom = "chevron")
```

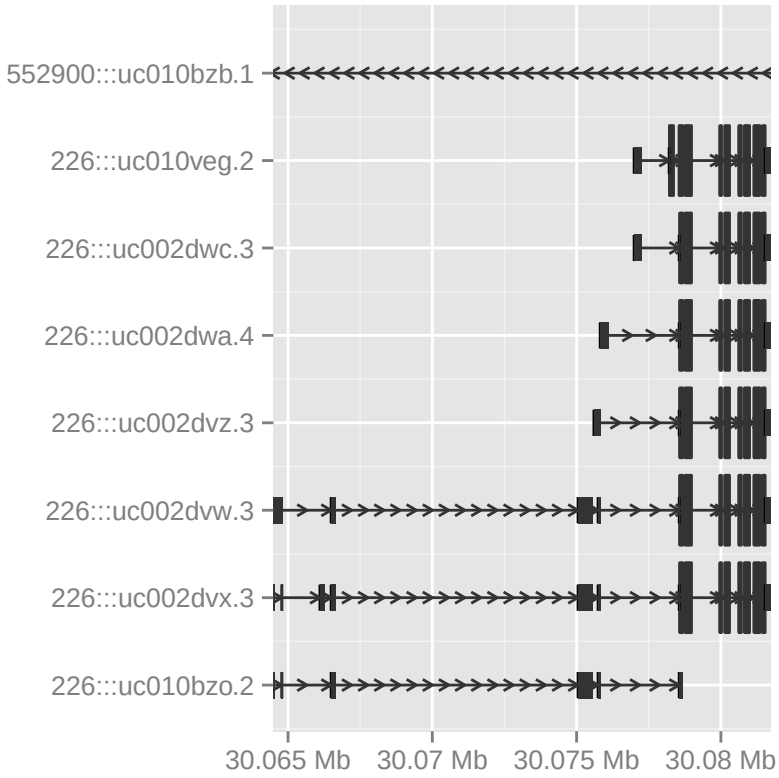



```
library(grid)
autoplot(txdb, which = aldoa.gr, arrow.rate = 0.001, length = unit(0.35, "cm"))
```



We also allow users to parse y labels from existing column in `TranscriptDb` object.

```
p <- autoplot(txdb, which = aldoa.gr, names.expr = "gene_id:::tx_name")
p
```


552900:::uc010bzb.1
226:::uc010veg.2
226:::uc002dwc.3
226:::uc002dwa.4
226:::uc002dvz.3
226:::uc002dvw.3
226:::uc002dvx.3
226:::uc010bzo.2
30.065 Mb
30.07 Mb
30.075 Mb
30.08 Mb

`scale_x_sequnit` is a add-on utility to revise the x-scale, it provides three unit

- **mb**: 1e6bp unit. default for autoplot,TranscriptDb.
- **kb**: 1e3bp unit.
- **bp**: 1bp unit

it's just post-graphic modification, won't re-load the parsing process. Figure

```
p + scale_x_sequnit("kb")
```

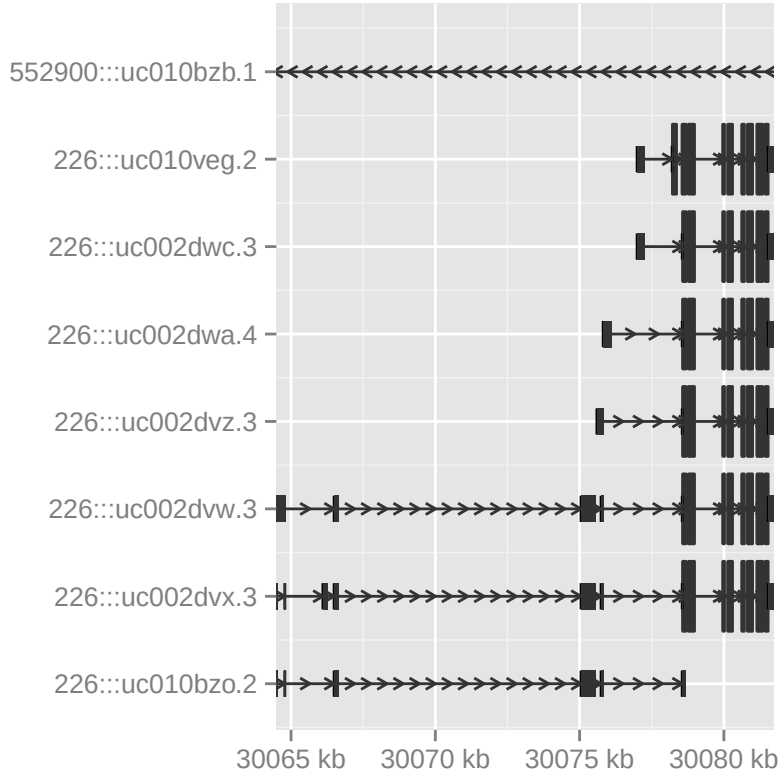


Figure 9.1: change the unit to kb.

9.2.2 geom_alignment

`stat_gene` is deprecated, and `geom_alignment` is the lower level API which facilitate construction layer by layer.

```
p1 <- ggplot() + geom_alignment(txdb, which = aldoa.gr)
```

Chapter 10

Visualize sequence

Chapter 11

Visualize matrix-related objects

Chapter 12

Visualize VCF files

Chapter 13

Visualize splicing events

Chapter 14

Miscellaneous

14.1 Themes

14.1.1 Plot theme

14.1.2 Track theme

14.2 Scales

Chapter 15

Session Information

```
sessionInfo()


## R version 3.0.1 (2013-05-16)
## Platform: x86_64-unknown-linux-gnu (64-bit)
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=C                 LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] grid      parallel  stats     graphics  grDevices utils
## [7] datasets  methods   base
##
## other attached packages:
##  [1] gridExtra_0.9.1
##  [2] biovizBase_1.8.1
##  [3] TxDb.Hsapiens.UCSC.hg19.knownGene_2.9.2
##  [4] GenomicFeatures_1.12.4
##  [5] AnnotationDbi_1.22.6
##  [6] Biobase_2.20.1
##  [7] rtracklayer_1.20.4
##  [8] ggbio_1.8.8
##  [9] ggplot2_0.9.3.1
## [10] GenomicRanges_1.12.5
## [11] IRanges_1.18.4
```

```
## [12] BiocGenerics_0.6.0
## [13] knitr_1.5.3
##
## loaded via a namespace (and not attached):
##  [1] BSgenome_1.28.0         Biostrings_2.28.0
##  [3] DBI_0.2-7               Hmisc_3.12-2
##  [5] MASS_7.3-29             RColorBrewer_1.0-5
##  [7] RCurl_1.95-4.1          RSQLite_0.11.4
##  [9] Rsamtools_1.12.4        VariantAnnotation_1.6.8
## [11] XML_3.98-1.1            biomaRt_2.16.0
## [13] bitops_1.0-6            cluster_1.14.4
## [15] colorspace_1.2-4        dichromat_2.0-0
## [17] digest_0.6.3            evaluate_0.5.1
## [19] formatR_0.9             gtable_0.1.2
## [21] highr_0.2.1             labeling_0.2
## [23] lattice_0.20-23         munsell_0.4.2
## [25] plyr_1.8                proto_0.3-10
## [27] reshape2_1.2.2          rpart_4.1-3
## [29] scales_0.2.3            stats4_3.0.1
## [31] stringr_0.6.2           tools_3.0.1
## [33] zlibbioc_1.6.0
```