biovizBase

March 24, 2012

`GCcontent` *GC content computation for BSgenome*

Description

Compute GC content in a certain region of a BSgenome object

Usage

```
GCcontent(obj, ..., view.width, as.prob = TRUE)
```

Arguments

`obj`	BSgenome object
`...`	Arguments passed to getSeq method for BSgenome package.
`view.width`	Passed to `letterFrequencyInSlidingView`, the constant (e.g. 35, 48, 1000) size of the "window" to slide along `obj`. The specified `letters` are tabulated in each window of length `view.width`. The rows of the result (see value) correspond to the various windows.
`as.prob`	If TRUE return percentage of GC content, otherwise return counts.

Details

GC content is an interesting variable may be related to various biological questions. So we need a way to compute GC content in a certain region of a reference genome. GCcontent function is a wrapper around getSeq function in BSgenome package and `letterFrequency`, `letterFrequencyInSlidingV` in Biostrings package.

if the `view.width` is specified, the GC content will be computed in the sliding view

Value

Numeric value indicate count or percentage

Author(s)

Tengfei Yin

Examples

```
library(BSgenome.Hsapiens.UCSC.hg19)
GCcontent(Hsapiens, GRanges("chr1", IRanges(1e6, 1e6 + 1000)))
```

`addSteppings-method`

Adding disjoint levels to a GenomicRanges object

Description

Adding disjoint levels to a GenomicRanges object

Usage

```
## S4 method for signature 'GenomicRanges'
addSteppings(obj, group.name, extend.size = 0)
```

Arguments

`obj`	A GenomicRanges object
`group.name`	Column name in the elementMetadata which specify the grouping information of all the entries. If provided, this will make sure all intervals belong to the same group will try to be on the same level and nothing falls in between.
`extend.size`	Adding invisible buffered region to the GenomicRanges object, if it's 10, then adding 5 at both end. This make the close neighbors assigned to the different levels and make your eyes easy to identify.

Details

This is a tricky question, for example, pair-end RNA-seq data could be represented as two set of GenomicRanges object, one indicates the read, one indicates the junction. At the same time, we need to make sure pair-ended read are shown on the same level, and nothing falls in between. For better visualization of the data, we may hope to add invisible extended buffer to the reads, so closely neighbored reads will be on the different levels.

Value

A modified GenmicRanges object with `.levels` as one column.

Author(s)

Tengfei Yin

Examples

```
library(GenomicRanges)
set.seed(1)
N <- 500
## sample GRanges
gr <- GRanges(seqnames =
              sample(c("chr1", "chr2", "chr3", "chrX", "chrY"),
                     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),
                group = sample(c("Normal", "Tumor"),
                  size = N, replace = TRUE),
                pair = sample(letters, size = N,
                  replace = TRUE))

## grouping and extending
head(addSteppings(gr))
head(addSteppings(gr, group.name = "pair"))
gr.close <- GRanges(c("chr1", "chr1"), IRanges(c(10, 20), width = 9))
addSteppings(gr.close)
addSteppings(gr.close, extend.size = 5)
```

`biovizBase-package` *biovizBase is a package which provides utilities and color scheme...*

Description

biovizBase is a package which provides utilities and color scheme for higher level graphic package which aim to visualize biological data especially genetic data.

Details

This package provides default color scheme for nucleotide, strand, amino acid, try to pass colorblind checking as possible as we can. And also provide giemsa stain result color scheme used to show cytoband. This package also provides utilites to manipulate and summarize raw data to get them ready to be visualized.

`colorBlindSafePal` *Color blind safe palette generator*

Description

This function help users create a function based on specified color blind safe palette. And the returned function could be used for color generation.

Usage

```
genDichromatPalInfo()
genBrewerBlindPalInfo()
genBlindPalInfo()
colorBlindSafePal(palette)
blind.pal.info
brewer.pal.blind.info
dichromat.pal.blind.info
```

Arguments

palette	A index numeric value or character. Please see blind.pal.info, the palette could be "pal_id" or names the row in which users could specify the palette you want to use.

Details

We get those color-blind safe palette based on http://colorbrewer2.org/ and http://geography.uoregon.edu/datagraphics/color_scales.htm those color are implemented in two packages, RColorBrewer and dichromat. But RColorBrewer doesn't provide subset of color-blind safe palette info. And dichromat doesn't group color based on "quality", "sequential" and "divergent", so we pick those color manually and create a combined palette, blind.pal.info.

colorBlindSafePal will return a function, this function will take two arguments, n and repeatable. if n is smaller than 3(n >= 3 is required by RColorBrewer), we use 3 instead and return a color vector. If n is over the maxcolors column in blind.pal.info, we use maxcolors instead when repeatable set to FALSE, if repeatable set to TRUE we repeat the color of all the allowed colors(length equals maxcolors) in the same order. This has special case in certain graphics which is always displayed side by side and don't worry about the repeated colors being neighbors.

genBrewerBlindPalInfo return brewer.pal.blind.info data frame containing all color-blind safe palettes from brewer.pal.info defined in RColorBrewer, but it's not only just subset of it, it also changes some maxcolors info.

genDichromatPalInfo return dichromat.pal.blind.info data frame.

genBlindPalInfo return blind.pal.info data frame.

Value

A color generating function with arguments n and repeatable. n specifying how many different discrete colors you want to get from them palette, and if repeatable turned on and set to TRUE, you can specify n even larger than maximum color. The color will be repeated following the same order.

Author(s)

Tengfei Yin

Examples

```
## Not run:
library(scales)
## brewer subse of only color blind palette
brewer.pal.blind.info
genBrewerBlindPalInfo()
## dichromat info
dichromat.pal.blind.info
genDichromatPalInfo()
## all color blind palette, adding id/pkg.
blind.pal.info
## with no parameters, just return blind.pal.info
colorBlindSafePal()
mypal <- colorBlindSafePal(20)
## or pass character name
mypal <- colorBlindSafePal("Set2")
mypal12 <- colorBlindSafePal(22)
show_col(mypal(12, repeatable = FALSE)) # warning
```

```
show_col(mypal(11, repeatable = TRUE))  # no warning, and repeat
show_col(mypal12(12))

## End(Not run)
```

`containLetters` *Checking if string contains letters or not*

Description

Test if a string contain any letters

Usage

```
containLetters(obj, all=FALSE)
```

Arguments

`obj`	String
`all`	If set to FALSE, return TRUE when any letters appears; if all is set to TRUE, return TRUE only when the string is composed of just letters.

Details

Useful when processing/sorting seqnames

Value

Logical value

Author(s)

tengfei

Examples

```
containLetters("XYZ123")
containLetters("XYZ123", TRUE)
```

darned_hg19_subset500

Subset of RNA editing sites in hg19...

Description

Subset of RNA editing sites in hg19

Usage

```
data(darned_hg19_subset500)
```

Details

This data set provides a subset(500 sites only) of hg19 RNA editing sites, and originally from DARNED http://darned.ucc.ie/ for the hg19 assembly.

Examples

```
data(darned_hg19_subset500)
darned_hg19_subset500
```

genesymbol

Gene symbols with position...

Description

Gene symbols with position

Usage

```
data(genesymbol)
```

Details

This data set provides genen symbols in human with position and starnd information, stored as GRanges object.

Examples

```
data(genesymbol)
head(genesymbol)
genesymbol["RBM17"]
```

`getBioColor` *Color scheme getter for biological data*

Description

This function tries to get default color scheme either from fixed palette or options for specified data set, usually just biological data.

Usage

```
getBioColor(type = c("DNA_BASES_N", "DNA_BASES", "DNA_ALPHABET",
                     "RNA_BASES_N", "RNA_BASES", "RNA_ALPHABET",
                     "IUPAC_CODE_MAP", "AMINO_ACID_CODE", "AA_ALPHABET",
                     "STRAND", "CYTOBAND"),
            source = c("option", "default"))
```

Arguments

`type`	Color set based on which you want to get.
`source`	"option" tries to get color scheme from `Options`. This allow user to edit the color globally. "default" gets fixed color scheme.

Details

Most data set specified in the type argument are defined in Biostrings package, such as "DNA_BASES", "DNA_ALPHABET", "RNA_BASES", "RNA_ALPHABET", "IUPAC_CODE_MAP", "AMINO_ACID_CODE", "AA_ALPHABET", please check the manual for more details.

"DNA_BASES_N" is just "DNA_BASES" with extra "N" used in certain cases, like the result from `applyPileup` in Rsamtools package. We start with the five most used nucleotides, `A,T,C,G,N`, In genetics, `GC-content` usually has special biological significance because GC pair is bound by three hydrogen bonds instead of two like AT pairs. So it has higher thermostability which could result in different significance, like higher annealing temperature in PCR. So we hope to choose warm color for `G,C` and cold color for `A,T`, and a color in between to represent `N`. They are chosen from diverging color set of `color brewer`. So we should be able to easily tell the GC enriched region. This set of color also passed vischeck for colorblind people.

In `GRanges` object, we have `strand` which contains three levels, +, -, *. We are using a qualitative color set from `Color Brewer`. This color set pass the colorblind test. It should be a safe color set to use to color strand.

For most default color scheme if they are under 18, we are trying to use package dichromat to set color for color blind people. But for data set that contains more than 18 objects, it's not possible to assign colorblind-safe color to them anymore. So we need to repeat some color. It should not matter too much, because even normal people cannot tell the difference anymore.

Here are the definition for the data sets.

DNA_BASES Contains `A,C,T,G`.

DNA_ALPHABET This alphabet contains all letters from the IUPAC Extended Genetic Alphabet (see "?IUPAC_CODE_MAP") + the gap ("-") and the hard masking ("+") letters.

DNA_BASES_N Contains `A,C,T,G,N`

RNA_BASES_N Contains `A,C,U,G,N`

RNA_BASES Contains `A,C,T,G`

RNA_ALPHABET This alphabet contains all letters from the IUPAC Extended Genetic Alphabet (see ?IUPAC_CODE_MAP) where "T" is replaced by "U" + the gap ("-") and the hard masking ("+") letters.

IUPAC_CODE_MAP The "IUPAC_CODE_MAP" named character vector contains the mapping from the IUPAC nucleotide ambiguity codes to their meaning.

AMINO_ACID_CODE Single-Letter Amino Acid Code (see "?AMINO_ACID_CODE").

AA_ALPHABET This alphabet contains all letters from the Single-Letter Amino Acid Code (see "?AMINO_ACID_CODE") + the stop ("*"), the gap ("-") and the hard masking ("+") letters

STRAND Contains `"+", "-", "*"`

CYTOBAND Contains giemsa stain results:`gneg, gpos25, gpos50, gpos75, gpos100, gvar, stalk, acen`. Color defined in package geneplotter.

Value

A character of vector contains color(rgb), and the names of the vector is originally from the name of different data set. e.g. for DNA_BASES, it's just `A,C,T,G`. This allow users to get color for a vector of specified names. Please see the examples below.

Author(s)

Tengfei Yin

Examples

```
opts <- getOption("biovizBase")
opts$DNABasesNColor[1] <- "red"
options(biovizBase = opts)
## get from option(default)
getBioColor("DNA_BASES_N")
## get default fixed color
getBioColor("DNA_BASES_N", source = "default")
seqs <- c("A", "C", "T", "G", "G", "G", "C")
## get colors for a sequence.
getBioColor("DNA_BASES_N")[seqs]
```

`getIdeogram` *Get ideogram.*

Description

Get ideogram w/o cytoband for certain genome

Usage

```
getIdeogram(genome, subchr, cytobands=TRUE)
```

Arguments

genome	Single specie names, which must be one of the result from `ucscGenomes()$db`. If missing, will invoke a menu for users to choose from.
subchr	A character vector used to subset the result.
cytobands	If TRUE, return ideogram with gieStain and name column. If FALSE, simply return the genome information for each chromosome.

Details

This function require a network connection, it will parse the data on the fly, function is a wrapper of some functionality from rtracklayer package to get certain table like cytoBand, a full table schema could be found http://genome.ucsc.edu/cgi-bin/hgTables in UCSC genome browser.This is useful for visualization of the whole genome or single chromosome, you can see some examples in ggbio package.

Value

A GRanges object.

Author(s)

Tengfei Yin

Examples

```
## Not run: hg19IdeogramCyto <- getIdeogram("hg19", cytoband = TRUE)
```

---

hg19Ideogram *Hg19 ideogram without cytoband information...*

Description

Hg19 ideogram without cytoband information

Usage

```
data(hg19Ideogram)
```

Details

This data set provides hg19 genome information wihout cytoband information.

Examples

```
data(hg19Ideogram)
hg19Ideogram
```

hg19IdeogramCyto *Hg19 ideogram with cytoband information...*

Description

Hg19 ideogram with cytoband information

Usage

```
data(hg19IdeogramCyto)
```

Details

This data set provides hg19 genome information with cytoband information.

Examples

```
data(hg19IdeogramCyto)
hg19IdeogramCyto
```

isIdeogram *Ideogram checking*

Description

Check if an object is ideogram or not

Usage

```
isIdeogram(obj)
```

Arguments

obj	object

Details

Simply test if it's the result coming from `getIdeogram` function or not, make sure it's a `GRanges` and with extra column

Value

A logical value to indicate it's a ideogram or not.

Author(s)

Tengfei Yin

Examples

```
data(hg19IdeogramCyto)
data(hg19Ideogram)
isIdeogram(hg19IdeogramCyto)
isIdeogram(hg19Ideogram)
```

---

isMatchedWithModel *Utils for Splicing Summary*

Description

Utilities used for summarizing isoforms

Usage

```
isJunctionRead(cigar)
isMatchedWithModel(model, gr)
```

Arguments

cigar	A CIGAR string vector.
model	A GRanges object.
gr	A GRanges object.

Details

`isJunctionRead` simply parsing the CIGAR string to see if there is "N" in between and return a logical vector of the same length as cigar parameters, indicate it's junction read or not.

`isMatchedWithModel` mapping `gr` to `model`, and counting overlapped cases for each row of model, If `gr` contains all the read, this will return a logical vector of the same length as `gr`, and indicate if each read is the support for this model. NOTICE: we only assume it's a full model, so each `model` here is simply one isoform. So we only treat the gaped reads which only overlapped with two consecutive exons in `model` as one support for it.

Value

Logical vectors.

Author(s)

Tengfei Yin

Examples

```
library(Rsamtools)
bamfile <- system.file("extdata", "SRR027894subRBM17.bam",
  package="biovizBase")
## get index of junction read
which(isJunctionRead(cigar(readBamGappedAlignments(bamfile))))
##
model <- GRanges("chr1", IRanges(c(10, 20, 30, 40), width  = 5))
gr <- GRanges("chr1", IRanges(c(10, 10, 12, 22, 33), c(31, 40, 22, 32,
```

```
  44)))
isMatchedWithModel(model, gr)
```

`isSimpleIdeogram` *Simple ideogram checking*

Description

Check if an object is a simple ideogram or not

Usage

```
isSimpleIdeogram(obj)
```

Arguments

`obj`	object

Details

test if it's GRanges or not, doesn't require cytoband information. But it double check to see if there is only one entry per chromosome

Value

A logical value to indicate it's a simple ideogram or not.

Author(s)

tengfei

Examples

```
data(hg19IdeogramCyto)
data(hg19Ideogram)
isSimpleIdeogram(hg19IdeogramCyto)
isSimpleIdeogram(hg19Ideogram)
```

`maxGap-method` *Estimated max gaps*

Description

Compute an estimated max gap information, which could be used as max.gap argument in `shringkageFun`.

Usage

```
## S4 method for signature 'GenomicRanges'
maxGap(obj, ratio = 0.0025)
```

Arguments

obj	GenomicRanges object
ratio	Multiple by the range of the provided gaps as the max gap.

Details

This function tries to estimate an appropriate max gap to be used for creating a shrinkage function.

Value

A numeric value

Author(s)

Tengfei Yin

See Also

shrinkageFun

Examples

```
require(GenomicRanges)
gr1 <- GRanges("chr1", IRanges(start = c(100, 300, 600),
end = c(200, 400, 800)))
gr2 <- GRanges("chr1", IRanges(start = c(100, 350, 550),
end = c(220, 500, 900)))
gaps.gr <- intersect(gaps(gr1, start = min(start(gr1))),
gaps(gr2, start = min(start(gr2))))
shrink.fun <- shrinkageFun(gaps.gr, max.gap = maxGap(gaps.gr))
shrink.fun(gr1)
shrink.fun(gr2)
```

---

pileupAsGRanges *Summarize reads for certain region*

Description

This function summarize reads from bam files for nucleotides on single base unit in a given region, this allows the downstream mismatch summary analysis.

Usage

```
pileupAsGRanges(bams, regions, DNABases=c("A", "C", "G", "T", "N"), ...)
```

Arguments

bams	A character which specify the bam file path.
regions	A GRanges object specifying the region to be summarized. This passed to which arguments in PileupParam.
DNABases	Nucleotide type you want to summarize in the result and in specified order. It must be one or more of A,C,G,T,N.
...	Extra parameters passed to PileupParam.

Details

It's a wrapper around `applyPileup` function in Rsamtools package, more detailed control could be found under manual of PileupParam function in Rsamtools. `pileupAsGRanges` function return a GRanges object which including summary of nucleotides, depth, bam file path. This object could be read directly into `pileupGRangesAsVariantTable` function for mismatch summary.

Value

A GRanges object, each row is one single base unit. and elementMetadata contains summary about this position about all nucleotides specified by DNABases. and `depth` for total reads, `bam` for file path.

Author(s)

Michael Lawrence, Tengfei Yin

Examples

```
## Not run:
library(Rsamtools)
data(genesymbol)
library(BSgenome.Hsapiens.UCSC.hg19)
bamfile <- system.file("extdata", "SRR027894subRBM17.bam", package="biovizBase")
test <- pileupAsGRanges(bamfile, region = genesymbol["RBM17"])
test.match <- pileupGRangesAsVariantTable(test, Hsapiens)
head(test[,-7])
head(test.match[,-5])

## End(Not run)
```

`pileupGRangesAsVariantTable` *Mismatch summary*

Description

Compare to reference genome and compute mismatch summary for certain region of reads.

Usage

```
pileupGRangesAsVariantTable(gr, genome, DNABases=c("A", "C", "G", "T", "N"))
```

Arguments

`gr`	A GRanges object, with nucleotides summary, each base take one column in elementMetadata or user can simply passed the returned result from `pileupAsGRanges` function to this function.
`genome`	BSgenome object, need to be the reference genome.
`DNABases`	Nucleotide types contained in passed GRanges object. Default is A/C/G/T/N, it tries to match the column names in elementMetadata to those default nucleotides. And treat the matched column as base names.

Details

User need to make sure to pass the right reference genome to this function to get the right summary. This function drop the position has no reads and only keep the region with coverage in the summary. The result could be used to show stacked barchart for mismatch summary.

Value

A GRanges object. Containing the following elementMetadata

- refNucleotide in reference genome.
- readNucleotide contained in the reads at particular position, if multiple nucleotide, either matched or unmatched are found, they will be summarized in different rows.
- countCount for read column.
- matchLogical value, whether matched to reference genome or not
- bamCharacter indicate bam file path.

Author(s)

Michael Lawrence, Tengfei Yin

Examples

```
## Not run:
library(Rsamtools)
data(genesymbol)
library(BSgenome.Hsapiens.UCSC.hg19)
bamfile <- system.file("extdata", "SRR027894subRBM17.bam", package="biovizBase")
test <- pileupAsGRanges(bamfile, region = genesymbol["RBM17"])
test.match <- pileupGRangesAsVariantTable(test, Hsapiens)
head(test[,-7])
head(test.match[,-5])

## End(Not run)
```

---

plotColorLegend *Show colors*

Description

Plot color legend, simple way to check your default color scheme

Usage

```
plotColorLegend(colors, labels, title)
```

Arguments

colors	A character vector of colors
labels	Labels to put aside colors, if missing, use names of the colors character vector.
title	Title for the color legend.

Details

Show color sheme as a legend style, labels

Value

A graphic device showing color legend.

Author(s)

Tengfei Yin

Examples

```
cols <- getOption("biovizBase")$baseColor
plotColorLegend(cols, title = "strand legend")
```

---

showColor *Show colors*

Description

Show colors with color string or names of color vectors.

Usage

```
showColor(colors, label = c("color", "name"))
```

Arguments

colors	A color character vector.
label	"color" label color with simple color string, and "name" label color with names of the vectors.

Value

A plot.

Author(s)

Tengfei Yin

Examples

```
## Not run:
showColor(getBioColor("CYTOBAND"))

## End(Not run)
```

shrinkageFun-method
Shrinkage function

Description

Create a shrinkage function based on specified gaps and shrinkage rate.

Usage

```
## For IRanges
## S4 method for signature 'IRanges'
shrinkageFun(obj, max.gap = 1L)

## For GenomicRanges
## S4 method for signature 'GenomicRanges'
shrinkageFun(obj, max.gap = 1L)
```

Arguments

obj	GenomicRanges object which represent gaps
max.gap	Gaps to be kept, it's a fixed value, if this value is bigger than certain gap interval, then that gap is not going to be shrunk.

Details

shrinkageFun function will read in a GenomicRanges object which represent the gaps, and return a function which works for another GenomicRanges object, to shrink that object based on previously specified gaps shrinking information. You could use this function to treat multiple tracks(e.g. GRanges) to make sure they shrunk based on the common gaps and the same ratio.

Value

A shrinkage function which could shrink a GenomicRanges object

Author(s)

Tengfei Yin

Examples

```
library(GenomicRanges)
gr1 <- GRanges("chr1", IRanges(start = c(100, 300, 600),
                               end = c(200, 400, 800)))
shrink.fun1 <- shrinkageFun(gaps(gr1), max.gap = maxGap(gaps(gr1), 0.15))
shrink.fun2 <- shrinkageFun(gaps(gr1), max.gap = 0)
shrink.fun1(gr1)
shrink.fun2(gr1)
```

transformGRangesForEvenSpace

Transform GRanges with New Coordinates

Description

For graphics, like linked plot, e.g. generated by `qplotRangesLinkedToData` function in package ggbio. we need to generate a new set of coordinates which is used for even spaced statistics track.

Usage

```
transformGRangesForEvenSpace(gr)
```

Arguments

`gr`	A GRanges object.

Details

Most used internally for special graphics, like `qplotRangesLinkedToData` function in package ggbio.

Value

A GRanges object as passed in, with new column x.new which indicate the static track coordinates, in this way, we could map the new coordinates with the old one.

Author(s)

Tengfei Yin

Examples

```
library(GenomicRanges)
gr <- GRanges("chr1", IRanges(seq(1,100, length.out = 10), width = 5))
transformGRangesForEvenSpace(gr)
```

transformGRangesToDfWithTicks

transformGRangesToDfWithTicks

Description

Transform GRanges object to a data.frame with ticks position stored in a list. This is different from simply coerce GRanges to a data frame, it is used for Grand Linear Compact View, which need to recompute the genomic coordinate and forcing them to be continued values.

Usage

```
transformGRangesToDfWithTicks(gr, fixed.length = NULL)
```

Arguments

gr	A GRanges object.
fixed.length	Fixed length for arranging space. If NULL, use data range, which means it's simply concatenate the data based on seqnames. If it's provided, it's required that it's a numeric vector and names of the vector corresponding to the seqnames.

Details

For compact grand linear view, we need to recompute the coordinates and concatenate them in a linear manner, remove the x axis labels and relabel them with chromosome names. So this function serve the basis for doing such transformation.

Value

A list contains a data frame which with adjusted coordinates and a vector indicates the ticks position, which will be used for Grand Linear Compact View labeling in some graphics.

Author(s)

Tengfei Yin

Examples

```
library(GenomicRanges)
gr <- GRanges("chr1", IRanges(seq(1,100, length.out = 10), width = 5))
transformGRangesToDfWithTicks(gr)
```

Index

∗Topic **datasets**
darned_hg19_subset500, 6
genesymbol, 6
hg19Ideogram, 9
hg19IdeogramCyto, 10

addSteppings
(addSteppings-method), 2
addSteppings,GenomicRanges-method
(addSteppings-method), 2
addSteppings-method, 2

biovizBase *(biovizBase-package)*, 3
biovizBase-package, 3
blind.pal.info
(colorBlindSafePal), 3
brewer.pal.blind.info
(colorBlindSafePal), 3

colorBlindSafePal, 3
containLetters, 5

darned_hg19_subset500, 6
dichromat.pal.blind.info
(colorBlindSafePal), 3

GCcontent, 1
genBlindPalInfo
(colorBlindSafePal), 3
genBrewerBlindPalInfo
(colorBlindSafePal), 3
genDichromatPalInfo
(colorBlindSafePal), 3
genesymbol, 6
getBioColor, 7
getIdeogram, 8

hg19Ideogram, 9
hg19IdeogramCyto, 10

isIdeogram, 10
isJunctionRead
(isMatchedWithModel), 11
isMatchedWithModel, 11
isSimpleIdeogram, 12

maxGap *(maxGap-method)*, 12
maxGap,GenomicRanges-method
(maxGap-method), 12
maxGap-method, 12

pileupAsGRanges, 13
pileupGRangesAsVariantTable, 14
plotColorLegend, 15

showColor, 16
shrinkageFun, *13*
shrinkageFun
(shrinkageFun-method), 17
shrinkageFun,GenomicRanges-method
(shrinkageFun-method), 17
shrinkageFun,IRanges-method
(shrinkageFun-method), 17
shrinkageFun-method, 17

transformGRangesForEvenSpace, 18
transformGRangesToDfWithTicks, 18