Package ‘varitas’

February 21, 2020

Type Package
Title Variant Calling in Targeted Analysis Sequencing Data
Version 0.0.1
Date 2020-02-07
Description Multi-caller variant analysis pipeline for targeted analysis sequencing (TAS) data. Features a modular, automated workflow that can start with raw reads and produces a user-friendly PDF summary and a spreadsheet containing consensus variant information.
SystemRequirements perl, bedtools (>=2.27.1), bwa
License GPL-2
Suggests testthat, knitr, markdown, futile.logger
Imports stringr, dplyr, yaml, openxlsx, VennDiagram, assertthat, magrittr, tools, utils, tidyr, doParallel, foreach
RoxygenNote 6.1.1
Encoding UTF-8
VignetteBuilder knitr
NeedsCompilation no
Author Adam Mills [aut, cre], Erle Holgersen [aut], Ros Cutts [aut], Syed Haider [aut]
Maintainer Adam Mills <Adam.Mills@icr.ac.uk>
Repository CRAN
Date/Publication 2020-02-21 12:00:03 UTC

R topics documented:

  add.option ................................................................. 4
  alternate.gene.sort ................................................................ 4
  build.variant.specification .................................................. 5
  caller.overlap.venn.diagram ................................................. 5
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>capitalize.caller</td>
<td>6</td>
</tr>
<tr>
<td>classify.variant</td>
<td>6</td>
</tr>
<tr>
<td>convert.ides.output</td>
<td>7</td>
</tr>
<tr>
<td>createdirectories</td>
<td>7</td>
</tr>
<tr>
<td>date.stamp.file.name</td>
<td>8</td>
</tr>
<tr>
<td>extract.sample.ids</td>
<td>8</td>
</tr>
<tr>
<td>filter.variant.file</td>
<td>9</td>
</tr>
<tr>
<td>filter.variants</td>
<td>9</td>
</tr>
<tr>
<td>fix.lofreq.af</td>
<td>10</td>
</tr>
<tr>
<td>fix.names</td>
<td>10</td>
</tr>
<tr>
<td>fix.varscan.af</td>
<td>11</td>
</tr>
<tr>
<td>get.base.substitution</td>
<td>11</td>
</tr>
<tr>
<td>get.bed.chromosomes</td>
<td>12</td>
</tr>
<tr>
<td>get.buildver</td>
<td>12</td>
</tr>
<tr>
<td>get.colours</td>
<td>13</td>
</tr>
<tr>
<td>get.coverage.by.amplicon</td>
<td>13</td>
</tr>
<tr>
<td>get.coverage.by.sample.statistics</td>
<td>14</td>
</tr>
<tr>
<td>get.fasta.chromosomes</td>
<td>14</td>
</tr>
<tr>
<td>get.file.path</td>
<td>15</td>
</tr>
<tr>
<td>get.filters</td>
<td>15</td>
</tr>
<tr>
<td>get.gene</td>
<td>16</td>
</tr>
<tr>
<td>get.miniseg.sample.files</td>
<td>16</td>
</tr>
<tr>
<td>get.option</td>
<td>17</td>
</tr>
<tr>
<td>get.panel.coverage.by.gene</td>
<td>17</td>
</tr>
<tr>
<td>get.pool.from.panel.data</td>
<td>18</td>
</tr>
<tr>
<td>get.varitas.options</td>
<td>18</td>
</tr>
<tr>
<td>get.vcf.chromosomes</td>
<td>19</td>
</tr>
<tr>
<td>in.varitas.options</td>
<td>19</td>
</tr>
<tr>
<td>logical.to.character</td>
<td>20</td>
</tr>
<tr>
<td>make.command.line.call</td>
<td>20</td>
</tr>
<tr>
<td>mean.field.value</td>
<td>21</td>
</tr>
<tr>
<td>merge.ides.annotation</td>
<td>21</td>
</tr>
<tr>
<td>merge.variants</td>
<td>22</td>
</tr>
<tr>
<td>overwrite.varitas.options</td>
<td>23</td>
</tr>
<tr>
<td>parse.job.dependencies</td>
<td>23</td>
</tr>
<tr>
<td>plot.amplicon.coverage.per.sample</td>
<td>24</td>
</tr>
<tr>
<td>plot.coverage.by.genome.order</td>
<td>24</td>
</tr>
<tr>
<td>plot.coverage.by.sample</td>
<td>25</td>
</tr>
<tr>
<td>plot.ontarget.percent</td>
<td>25</td>
</tr>
<tr>
<td>plot.paired.percent</td>
<td>26</td>
</tr>
<tr>
<td>post.processing</td>
<td>26</td>
</tr>
<tr>
<td>prepare.bam.specification</td>
<td>27</td>
</tr>
<tr>
<td>prepare.fastq.specification</td>
<td>28</td>
</tr>
<tr>
<td>prepare.miniseg.specifications</td>
<td>29</td>
</tr>
<tr>
<td>prepare.vcf.specification</td>
<td>30</td>
</tr>
<tr>
<td>process.coverage.reports</td>
<td>30</td>
</tr>
<tr>
<td>process.sample.contamination.checks</td>
<td>31</td>
</tr>
<tr>
<td>process.total.coverage.statistics</td>
<td>31</td>
</tr>
</tbody>
</table>
topics documented:

- read.all.calls ........................................................... 32
- read.ides.file .......................................................... 32
- read.variant.calls ..................................................... 33
- read.yaml ................................................................. 33
- run.alignment ............................................................. 34
- run.alignment.sample .................................................. 35
- run.all.scripts .......................................................... 36
- run.annotation ........................................................... 36
- run.annovar.vcf ......................................................... 37
- run.filtering.txt ......................................................... 38
- run.ides ................................................................. 39
- run.lofreq.sample ....................................................... 40
- run.muse.sample ......................................................... 41
- run.mutect.sample ...................................................... 42
- run.post.processing .................................................... 43
- run.target.qc ............................................................ 44
- run.target.qc.sample ................................................... 45
- run.vardict.sample ..................................................... 45
- run.variant.calling ..................................................... 46
- run.varitas.pipeline ................................................... 48
- run.varitas.pipeline.hybrid ........................................ 49
- run.varscan.sample .................................................... 50
- save.config .............................................................. 51
- save.coverage.excel ................................................... 52
- save.variants.excel .................................................... 52
- set.varitas.options ..................................................... 53
- split.on.column ........................................................ 53
- sum.dp4 ................................................................. 54
- system.ls ............................................................... 54
- tabular.mean ............................................................ 55
- tabular.median .......................................................... 55
- trinucleotide.barplot .................................................. 56
- variant.recurrence.barplot ......................................... 56
- variants.caller.barplot ................................................. 57
- variants.sample.barplot ............................................... 57
- verify.bam.specification ............................................... 58
- verify.bwa.index ....................................................... 58
- verify.fasta.index ..................................................... 59
- verify.fastq.specification ............................................ 59
- verify.sequence.dictionary ......................................... 60
- verify.varitas.options ................................................ 60
- verify.vcf.specification .............................................. 61

Index 62
add.option

Description
Add option to nested list of options. Applied recursively

Usage
add.option(name, value, old.options, nesting.character = "\.")

Arguments
name Option name. Nesting is indicated by character specified in nesting.character.
value New value of option
old.options Nested list the option should be added to
nesting.character String giving Regex pattern of nesting indication string. Defaults to \\.

Value
Nested list with updated options

alternate.gene.sort

Description
Given a data frame containing coverage statistics and gene information, returns that frame with the rows sorted by alternating gene size (for plotting)

Usage
alternate.gene.sort(coverage.statistics)

Arguments
coverage.statistics Data frame of coverage statistics

Details
Genes have varying numbers of associated amplicons and when plotting coverage statistics, if two genes with very low numbers of amplicons are next to each other, the labels will overlap. This function sorts the coverage statistics data frame in a way that places the genes with the most amplicons (largest) next to those with the least (smallest).
**build.variant.specification**

**Value**

Coverage statistics data frame sorted by alternating gene size

**Description**

Build data frame with paths to variant files.

**Usage**

```
build.variant.specification(sample.ids, project.directory)
```

**Arguments**

- `sample.ids` Vector of sample IDs. Must match subdirectories in `project.directory`.
- `project.directory` Path to directory where sample subdirectories

**Details**

Parses through sample IDs in a project directory and returns paths to variant files based on (theoretical) file name patterns. Useful for testing, or for entering the pipeline at non-traditional stages.

**Value**

Data frame with paths to variant files.

**caller.overlap.venn.diagram**

. Make Venn diagram of variant caller overlap

**Description**

. Make Venn diagram of variant caller overlap

**Usage**

```
caller.overlap.venn.diagram(variants, file.name)
```

**Arguments**

- `variants` Data frame containing variants, typically from `merge.variants` function
- `file.name` Name of output file
**capitalize.caller**

**Description**

Capitalize variant caller name

**Usage**

```r
capitalize.caller(caller)
capitalise.caller(caller)
```

**Arguments**

- `caller`: Character vector of callers to be capitalized

**Value**

Vector of same length as caller where eligible callers have been capitalized

---

**classify.variant**

**Description**

Classify a variant as SNV, MNV, or indel based on the reference and alternative alleles

**Usage**

```r
classify.variant(ref, alt)
```

**Arguments**

- `ref`: Vector of reference bases
- `alt`: Vector of alternate bases

**Value**

Character vector giving type of variant.
convert.ides.output

Convert output of iDES step 1 to variant call format

Usage

convert.ides.output(filename, output = TRUE, 
output.suffix = ".calls.txt", minreads = 5, mindepth = 50)

Arguments

filename Path to file  
output Logical indicating whether output should be saved to file. Defaults to true.  
output.suffix Suffix to be appended to input filename if saving results to file  
minreads Minimum numbers of reads  
mindepth Minimum depth

Value

potential.calls Data frame of converted iDES calls

create.directories

create.directories

Description

Create directories in a given path

Usage

create.directories(directory.names, path)

Arguments

directory.names Vector of names of directories to be created  
path Path where directories should be created
**date.stamp.file.name**

**Description**

Prefix file name with a date-stamp.

**Usage**

```r
date.stamp.file.name(file.name, date = Sys.Date(), separator = "_")
```

**Arguments**

- `file.name`: File name to be date-stamped
- `date`: Date to be added. Defaults to current date.
- `separator`: String that should separate the date from the file name. Defaults to a single underscore.

**Value**

String giving the datestamped file name

**Examples**

```r
date.stamp.file.name('plot.png');
date.stamp.file.name('yesterdays_plot.png', date = Sys.Date() - 1);
```

---

**extract.sample.ids**

**Description**

Extract sample IDs from a set of paths to files in sample-specific subfolders

**Usage**

```r
extract.sample.ids(paths, from.filename = FALSE)
```

**Arguments**

- `paths`: vector of file paths
- `from.filename`: Logical indicating whether sample ID should be extracted from filename rather than path

**Value**

vector of extracted sample IDs
filter.variant.file  Filter variants in file.

Description
Filter variants from file, and save to output. Wrapper function that opens the variant file, calls filter.variants, and saves the result to file.

Usage
filter.variant.file(variant.file, output.file, config.file = NULL, caller = c("vardict", "ides", "mutect", "pgm", "consensus"))

Arguments
- variant.file: Path to variant file
- output.file: Path to output file
- config.file: Path to config file to be used. If not supplied, will use the pre-existing VariTAS options.
- caller: Name of caller used (needed to match appropriate filters from settings)

Value
None

filter.variants  Filter variant calls

Description
Filter data frame of variant calls based on thresholds specified in settings.

Usage
filter.variants(variants, caller = c("vardict", "ides", "mutect", "pgm", "consensus", "isis", "varscan", "lofreq"), config.file = NULL, verbose = FALSE)

Arguments
- variants: Data frame of variant calls with ANNOVAR annotation, or path to variant file.
- caller: Name of caller used (needed to match appropriate filters from settings)
- config.file: Path to config file to be used. If not supplied, will use the pre-existing VariTAS options.
- verbose: Logical indicating whether to output descriptions of filtering steps. Defaults to False, useful for debugging.
Value

filtered.variants Data frame of filtered variants

fix.lofreq.af  fix.lofreq.af

Description

LoFreq also does not output allele frequencies, so this script calculates them from the DP (depth) and AD (variant allele depth) values—which are also not output nicely—and adds them to the annotated vcf.

Usage

fix.lofreq.af(variant.specification)

Arguments

variant.specification
  Data frame of variant file information

fix.names  Fix variant call column names

Description

Fix headers of variant calls to prepare for merging. This mostly consists in making sure the column headers will be unique by prefixing the variant caller in question.

Usage

fix.names(column.names, variant.caller, sample.id = NULL)

Arguments

column.names  Character vector of column names
variant.caller  String giving name of variant caller
sample.id  Optional sample ID. Used to fix headers.

Value

new.column.names Vector of column names after fixing]
Description

VarScan does not output allele frequencies, so this script calculates them from the DP (depth) and AD (variant allele depth) values and adds them to the annotated vcf.

Usage

fix.varscan.af(variant.specification)

Arguments

variant.specification

Data frame of variant file information

get.base.substitution  Get base substitution

Description

Get base substitution represented by pyrimidine in base pair. If more than one base in REF/ALT (i.e. MNV or indel rather than SNV), NA will be returned

Usage

generate.base.substitution(ref, alt)

Arguments

ref  Vector of reference bases
alt  Vector of alternate bases

Value

base.substitutions
get.bed.chromosomes

Description
Extract chromosomes from bed file

Usage
get.bed.chromosomes(bed)

Arguments
bed Path to BED file

Value
Vector containing all chromosomes in BED file

g.get.buildver

Description
Get build version (hg19/hg38) based on settings.
Parses VariTAS pipeline settings to get the build version. When this function was first developed, the idea was to be able to explicitly set ANNOVAR filenames based on the build version.

Usage
g.get.buildver()

Value
String giving reference genome build version (hg19 or hg38)
**get.colours**

*Generate a colour scheme*

**Description**

Generate a colour scheme

**Usage**

get.colours(n)

**Arguments**

- **n**
  - Number of colours desired

**Value**

Colour.scheme generated colours

---

**get.coverage.by.amplicon**

*Process sample coverage per amplicon data*

**Description**

Parse coverageBed output to get coverage by amplicon

**Usage**

get.coverage.by.amplicon(project.directory)

**Arguments**

- **project.directory**
  - Path to project directory. Each sample should have its own subdirectory

**Value**

combined.data Data frame giving coverage per amplicon per sample.

**References**

http://bedtools.readthedocs.io/en/latest/content/tools/coverage.html
get.coverage.by.sample.statistics

*Get statistics about coverage per sample*

**Description**

Get statistics about coverage per sample

**Usage**

get.coverage.by.sample.statistics(project.directory)

**Arguments**

- **project.directory**
  Path to project directory. Each sample should have its own subdirectory

**Value**

coverage.by.sample.statistics Data frame with coverage statistics per sample

---

get.fasta.chromosomes

g**et.fasta.chromosomes**

**Description**

Extract chromosomes from fasta headers.

**Usage**

get.fasta.chromosomes(fasta)

**Arguments**

- **fasta**
  Path to reference fasta

**Value**

Vector containing all chromosomes in fasta file.
get.file.path

Description

Get absolute path to sample-specific file for one or more samples

Usage

get.file.path(sample.ids, directory, extension = NULL,
          allow.multiple = FALSE, allow.none = FALSE)

Arguments

  sample.ids  Vector of sample IDs to match filename on
  directory   Path to directory containing files
  extension   String giving extension of file
  allow.multiple Boolean indicating whether to allow multiple matching files. Defaults to false, which throws an error if the query matches more than one file.
  allow.none  Boolean indicating whether to allow no matching files. Defaults to false, which throws an error if the query does not match any files.

Value

Paths to matched files

get.filters

Description

Determine filters per caller, given default and caller-specific values.

Usage

get.filters(filters)

Arguments

  filters  List of filter values. These will be updated to use default as the baseline, with caller-specific filters taking precedence if supplied.

Value

A list with updated filters
get.gene

Description

Use guesswork to extract gene from data frame of targeted panel data. The panel designer output can change, so try to guess what the format is.

Usage

get.gene(bed.data)

Arguments

bed.data Data frame containing data from bed file

Value

vector of gene names, one entry for each row of bed.data

get.miniseq.sample.files

Description

Get files for a sample in a directory, ensuring there’s only a single match per sample ID.

Usage

get.miniseq.sample.files(sample.ids, directory, file.suffix = "_S\d{1,2}_.*")

Arguments

sample.ids Vector of sample ids. Should form first part of file name
directory Directory where files can be found
file.suffix Regex expression for end of file name. For example, ‘file.suffix = ’_S\d1,2_*R1_*’’ will match R1 files.

Value

Character vector of file paths
get.option

Helper function to recursively get an VariTAS option

Description
Helper function to recursively get an VariTAS option

Usage
get.option(name, varitas.options = NULL, nesting.character = "\"\")

Arguments
name Option name
varitas.options Optional list of options to search in
nesting.character String giving Regex pattern of nesting indication string. Defaults to '\'

Value
value Requested option

get.panel.coverage.by.gene

Summarise panel coverage by gene

Description
Summarise panel coverage by gene

Usage
get.panel.coverage.by.gene(panel.file, gene.col = 5)

Arguments
panel.file path to panel
gene.col index of column containing gene name

Value
panel.coverage.by.gene data frame giving the number of amplicons and their total length by gene
get.varitas.options

get.pool.from.panel.data

*Get pool corresponding to each amplicon*

**Description**

The bed files are not consistent, so it’s not clear where the pool will appear. This function parses through the columns to identify where the pool

**Usage**

```
get.pool.from.panel.data(panel.data)
```

**Arguments**

- `panel.data` data frame pool should be extracted from

**Value**

- `pools` vector of pool information

get.varitas.options

*Return VariTAS settings*

**Description**

Return VariTAS settings

**Usage**

```
get.varitas.options(option.name = NULL, nesting.character = "\"\")
```

**Arguments**

- `option.name` Optional name of option. If no name is supplied, the full list of VariTAS options will be provided.
- `nesting.character` String giving Regex pattern of nesting indication string. Defaults to ‘\\’

**Value**

- `varitas.options` list specifying VariTAS options

**Examples**

```
reference.build <- get.varitas.options('reference_build');
mutect.filters <- get.varitas.options('filters.mutect');
```
### Description
Extract chromosomes from a VCF file.

### Usage
```
get.vcf.chromosomes(vcf)
```

### Arguments
- **vcf**
  Path to VCF file

### Value
Vector containing all chromosomes in VCF

---

### in.varitas.options
*Check if a key is in VariTAS options*

### Description
Check if a key is in VariTAS options

### Usage
```
in.varitas.options(option.name = NULL, varitas.options = NULL, nesting.character = "\"\")
```

### Arguments
- **option.name**
  String giving name of option (with different levels joined by `nesting.character`)
- **varitas.options**
  Ampliseq options as a list. If missing, they will be obtained from `get.varitas.options()`
- **nesting.character**
  String giving Regex pattern of nesting indication string. Defaults to '\'

### Value
in.options Boolean indicating if the option name exists in the current varitas options
logical.to.character  

**Description**  
Convert a logical vector to a T/F coded character vector. Useful for preventing unwanted T->TRUE nucleotide conversions  

**Usage**  
logical.to.character(x)  

**Arguments**  
x  
Vector to be converted  

**Value**  
Character vector after converting TRUE/FALSE  

make.command.line.call  

**Description**  
Make string with command line call from its individual components  

**Usage**  
make.command.line.call(main.command, options = NULL, flags = NULL, option.prefix = "--", option.separator = " ", flag.prefix = "--")  

**Arguments**  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>main.command</td>
<td>String or vector of strings giving main part of command (e.g. &quot;python test.py&quot; or c(&quot;python&quot;, &quot;test.py&quot;))</td>
</tr>
<tr>
<td>options</td>
<td>Named vector or list giving options</td>
</tr>
<tr>
<td>flags</td>
<td>Vector giving flags to include.</td>
</tr>
<tr>
<td>option.prefix</td>
<td>String to preface all options. Defaults to &quot;--&quot;</td>
</tr>
<tr>
<td>option.separator</td>
<td>String to separate options form their values. Defaults to a single space.</td>
</tr>
<tr>
<td>flag.prefix</td>
<td>String to preface all flags. Defaults to &quot;--&quot;</td>
</tr>
</tbody>
</table>

**Value**  
command string giving command line call
### mean.field.value

#### Description

Get mean value of a variant annotation field

#### Usage

```r
## S3 method for class 'field.value'
mean(variants, field = c("TUMOUR.DP", "NORMAL.DP", "NORMAL.AF", "TUMOUR.AF", "QUAL"),
     caller = c("consensus", "vardict", "pgm", "mutect", "isis", "varscan", "lofreq"))
```

#### Arguments

- `variants`: Data frame with variants
- `field`: String giving field of interest.
- `caller`: String giving caller to calculate values from

#### Details

As part of the variant merging process, annotated variant data frames are merged into one, with the value from each caller prefixed by CALLER. For example, the VarDict normal allele frequency will have header VARDICT.NORMAL.AF. This function takes the average of all callers’ value for a given field, removing NA’s. If only a single caller is present in the data frame, that value is returned.

#### Value

Vector of mean values.

---

### merge.ides.annotation

#### Description

Merge potential iDES calls with variant annotation.

#### Usage

```r
## S3 method for class 'ides.annotation'
merge(ides.filename, output = TRUE,
      output.suffix = ".ann.txt",
      annotor.suffix.pattern = ".annotor.hg(\d)\_multianno.txt")
```
merge.variants

Arguments

ides.filename Path to formatted iDES output (typically from convert.ides.output file)
output Logical indicating whether output should be saved to file. Defaults to true.
output.suffix Suffix to be appended to input filename if saving results to file
annovar.suffix.pattern Suffix to match ANNOVAR file

Details

The VarDict variant calling includes a GATK call merging the call vcf file (allele frequency information etc.) with the ANNOVAR annotation, and saving the result as a table. This function is an attempt to emulate that step for the iDES calls.

Value

annotated.calls Data frame of annotations and iDES output.

merge.variants Merge variants

Description

Merge variants from multiple callers and return a data frame of merged calls. By default filtering is also applied, although this behaviour can be turned off by setting apply.filters to FALSE.

Usage

## S3 method for class 'variants'
merge(variant.specification, apply.filters = TRUE,
      remove.structural.variants = TRUE,
      separate.consensus.filters = FALSE, verbose = FALSE)

Arguments

variant.specification Data frame containing details of file paths, sample IDs, and caller.
apply.filters Logical indicating whether to apply filters. Defaults to TRUE.
remove.structural.variants Logical indicating whether structural variants (including CNVs) should be removed. Defaults to TRUE.
separate.consensus.filters Logical indicating whether to apply different thresholds to variants called by more than one caller (specified under consensus in config file). Defaults to FALSE.
verbose Logical indicating whether to print information to screen
**overwrite.varitas.options**

**Value**

Data frame

---

**Description**

Overwrite VariTAS options with options provided in config file.

**Usage**

`overwrite.varitas.options(config.file)`

**Arguments**

- `config.file` Path to config file that should be used to overwrite options

**Value**

None

**Examples**

```r
## Not run:
config <- file.path(path.package("varitas"), "config.yaml")
overwrite.varitas.options(config)

## End(Not run)
```

---

**parse.job.dependencies**

**Parse job dependencies**

---

**Description**

Parse job dependencies to make the functions more robust to alternate inputs (e.g. people writing alignment instead of bwa)

**Usage**

`parse.job.dependencies(dependencies)`
plot.coverage.by.genome.order

Arguments
dependencies  Job dependency strings to be parsed.

Value
parsed.dependencies  Vector of job dependencies after reformatting.

plot.amplicon.coverage.per.sample

Description
Create one scatterplot per sample, showing coverage per amplicon, and an additional plot giving the median

Usage
## S3 method for class 'amplicon.coverage.per.sample'
plot(coverage.statistics,
     output.directory)

Arguments
coverage.statistics  Data frame containing coverage per amplicon per sample, typically from get.coverage.by.amplicon.
output.directory  Directory where per sample plots should be saved

Value
None

plot.coverage.by.genome.order

Description
Plot amplicon coverage by genome order

Usage
## S3 method for class 'coverage.by.genome.order'
plot(coverage.data)
plot.coverage.by.sample

**Arguments**

- `coverage.data` data frame with results from bedtools coverage command

---

plot.coverage.by.sample

**Description**

Make a barplot of coverage per sample

**Usage**

```r
## S3 method for class 'coverage.by.sample'
plot(coverage.sample, file.name, 
    statistic = c("mean", "median"))
```

**Arguments**

- `coverage.sample` Data frame of coverage data, typically from `get.coverage.by.sample.statistics`
- `file.name` Name of output file
- `statistic` Statistic to be plotted (mean or median)

**Value**

None

---

plot.ontarget.percent

**Description**

Make a scatterplot of ontarget percent per sample

**Usage**

```r
## S3 method for class 'ontarget.percent'
plot(coverage.sample, file.name)
```

**Arguments**

- `coverage.sample` Data frame of coverage data, typically from `get.coverage.by.sample.statistics`
- `file.name` Name of output file
Value
None

plot.paired.percent  plot.paired.percent

Description
Make a barplot of percent paired reads per sample

Usage
## S3 method for class 'paired.percent'
plot(coverage.sample, file.name)

Arguments
coverage.sample
Data frame of coverage data, typically from get.coverage.by.sample.statistics
file.name
Name of output file

Value
None

post.processing  Post-processing of variants to generate outputs

Description
Post-processing of variants to generate outputs

Usage
post.processing(variant.specification, project.directory, 
config.file = NULL, variant.callers = NULL, 
remove.structural.variants = TRUE, 
separate.consensus.filtres = FALSE, sleep = FALSE, verbose = FALSE)
Arguments

variant.specification
Data frame specifying variants to be processed, or path to data frame (useful if calling from Perl)

project.directory
Directory where output should be stored. Output files will be saved to a dates-tamped subdirectory

config.file
Path to config file specifying post-processing options. If not provided, the current options are used (i.e. from get.varitas.options())

variant.callers
Optional vector of variant callers for which filters should be included in Excel file

remove.structural.variants
Logical indicating whether structural variants (including CNVs) should be removed. Defaults to TRUE.

separate.consensus.filters
Logical indicating whether to apply different thresholds to variants called by more than one caller (specified under consensus in config file). Defaults to FALSE.

sleep
Logical indicating whether script should sleep for 60 seconds before starting.

verbose
Logical indicating whether to print verbose output

Value

None

prepare.bam.specification

Prepare Bam specification data frame to standardized format for downstream analyses.

Description

This function prepares a data frame that can be used to run variant callers. For matched normal variant calling, this data frame will contain three columns with names: sample.id, tumour.bam, normal.bam For unpaired variant calling, the data frame will contain two columns with names: sample.id, tumour.bam

Usage

prepare.bam.specification(sample.details, paired = TRUE,
    sample.id.column = 1, tumour.bam.column = 2, normal.bam.column = 3)
Arguments

sample.details Data frame where each row represents a sample to be run. Must contain sample ID, path to tumour BAM, and path to normal BAM.

paired Logical indicating whether the sample specification is for a paired analysis.

sample.id.column Index or string giving column of sample.details that contains the sample ID

tumour.bam.column Index or string giving column of sample.details that contains the path to the tumour BAM

normal.bam.column Index or string giving column of sample.details that contains the path to the normal BAM

Value

bam.specification Data frame with one row per sample to be run

Description

Prepare FASTQ specification data frame to standardized format for downstream analyses.

Usage

prepare.fastq.specification(sample.details, sample.id.column = 1, fastq.columns = c(2, 3), patient.id.column = NA, tissue.column = NA)

Arguments

sample.details Data frame where each row represents a sample to be run. Must contain sample ID, path to tumour BAM, and path to normal BAM.

sample.id.column Index or string giving column of sample.details that contains the sample ID

fastq.columns Index or string giving column(s) of sample.details that contain path to FASTQ files

patient.id.column Index or string giving column of sample.details that contains the patient ID
tissue.column Index or string giving column of sample.details that contains information on tissue (tumour/normal)
**Details**

This function prepares a data frame that can be used to run alignment. For paired-end reads, this data frame will contain three columns with names: sample.id, reads, mates. For single-end reads, the data frame will contain two columns with names: sample.id, reads.

**Value**

Data frame with one row per sample to be run

---

**Description**

Process a MiniSeq directory and sample sheet to get specification data frames that can be used to run the VariTAS pipeline.

Note: This assumes normal samples are not available.

**Usage**

```r
prepare.miniseq.specifications(sample.sheet, miniseq.directory)
```

**Arguments**

- `sample.sheet`: Data frame containing sample information, or path to a MiniSeq sample sheet
- `miniseq.directory`: Path to directory with MiniSeq files

**Value**

A list with specification data frames 'fastq', 'bam', and 'vcf' (as applicable)

**Examples**

```r
miniseq.sheet <- file.path(path.package('varitas'), 'extdata/miniseq/Example_template.csv')
miniseq.directory <- file.path(path.package('varitas'), 'extdata/miniseq')
miniseq.info <- prepare.miniseq.specifications(miniseq.sheet, miniseq.directory)
```
**prepare.vcf.specification**

**Description**
Prepare VCF specification data frame for annotation

**Usage**

```r
prepare.vcf.specification(vcf.details, sample.id.column = 1,
                         vcf.column = 2, job.dependency.column = NA, caller.column = NA)
```

**Arguments**
- `vcf.details`: Data frame containing details of VCF files
- `sample.id.column`: Identifier of column in `vcf.details` containing sample IDs (index or name)
- `vcf.column`: Identifier of column in `vcf.details` containing VCF file (index or name)
- `job.dependency.column`: Identifier of column in `vcf.details` containing job dependency (index or name)
- `caller.column`: Identifier of column in `vcf.details` containing caller (index or name)

**Value**
Properly formatted VCF details

**process.coverage.reports**

*Process coverageBed reports*

**Description**
Process the coverage reports generated by bedtools coverage tool.

**Usage**

```r
process.coverage.reports(project.directory)
```

**Arguments**
- `project.directory`: Path to project directory. Each sample should have its own subdirectory

**Value**
final.statistics data frame of coverage statistics generated by parsing through coverage reports
**process.sample.contamination.checks**

*Process sample contamination checks*

**Description**

Takes *selfSM reports generated by VerifyBamID during alignment, and returns a vector of freemix scores. The freemix score is a sequence only estimate of sample contamination that ranges from 0 to 1.

Note: Targeted panels are often too small for this step to work properly.

**Usage**

```
process.sample.contamination.checks(project.directory)
```

**Arguments**

- **project.directory**
  
  Path to project directory. Each sample should have its own subdirectory

**Value**

- freemix.scores Data frame giving sample contamination (column freemix) score per sample.

**References**

[https://genome.sph.umich.edu/wiki/VerifyBamID](https://genome.sph.umich.edu/wiki/VerifyBamID)

---

**process.total.coverage.statistics**

*Process total coverage statistics*

**Description**

Process reports generated by flagstat. Assumes reports for before and after off-target filtering have been written to the same file, with separating headers

**Usage**

```
process.total.coverage.statistics(project.directory)
```

**Arguments**

- **project.directory**
  
  Path to project directory. Each sample should have its own subdirectory
Value

data frame with extracted statistics

---

**read.all.calls**

**Description**

Read all calls made with a certain caller

**Usage**

```r
read.all.calls(sample.ids, caller = c("vardict", "mutect", "pgm"),
project.directory, patient.ids = NULL, apply.filters = TRUE,
variant.file.pattern = NULL)
```

**Arguments**

- `sample.ids`: Vector giving sample IDs to process
- `caller`: String indicating which caller was used
- `project.directory`: Path to project directory
- `patient.ids`: Optional vector giving patient ID (or other group) corresponding to each sample
- `apply.filters`: Logical indicating whether filters specified in VariTAS options should be applied. Defaults to TRUE.
- `variant.file.pattern`: Pattern indicating where the variant file can be found. Sample ID should be indicated by `SAMPLE_ID`

**Value**

`combined.variant.calls` Data frame with variant calls from all patients

---

**read.ides.file**

**Description**

Read iDES output

**Usage**

```r
read.ides.file(filename)
```
**read.variant.calls**

**Arguments**
- filename: path to file

**Value**
- ides.data: data frame read from iDES output

**Description**
Read variant calls from file and format for ease of downstream analyses.

**Usage**

```r
read.variant.calls(variant.file, variant.caller)
```

**Arguments**
- variant.file: Path to variant file.
- variant.caller: String indicating which variant caller was used. Needed to format the headers.

**Value**
- variant.calls: Data frame of variant calls

**read.yaml**

**Description**
Read a yaml file.

**Usage**

```r
read.yaml(file.name)
```

**Arguments**
- file.name: Path to yaml file

**Value**
- list containing contents of yaml file
run.alignment

Run alignment

Description

Run alignment

Usage

run.alignment(fastq.specification, output.directory, paired.end = FALSE, sample.directories = TRUE, output.subdirectory = FALSE, job.name.prefix = NULL, job.group = "alignment", quiet = FALSE, verify.options = !quiet)

Arguments

fastq.specification
   Data frame detailing FASTQ files to be processed, typically from prepare.fastq.specification
output.directory
   Path to project directory
paired.end
   Logical indicating whether paired-end sequencing was performed
sampledirectories
   Logical indicating whether all sample files should be saved to sample-specific subdirectories (will be created)
output.subdirectory
   If further nesting is required, name of subdirectory. If no further nesting, set to FALSE
job.name.prefix
   Prefix for job names on the cluster
job.group
   Group job should be associated with on cluster
quiet
   Logical indicating whether to print commands to screen rather than submit them
verify.options
   Logical indicating whether to run verify.varitas.options

Details

Runs alignment (and related processing steps) on each sample.

Value

None
**Examples**

```r
run.alignment(
  fastq.specification = data.frame(
    sample.id = c('1', '2'),
    reads = c('1-R1.fastq.gz', '2-R1.fastq.gz'),
    mates = c('1-R2.fastq.gz', '2-R2.fastq.gz'),
    patient.id = c('P1', 'P1'),
    tissue = c('tumour', 'normal')
  ),
  output.directory = '.','
  quiet = TRUE,
  paired.end = TRUE
)
```

---

**run.alignment.sample**  
*Run alignment for a single sample*

**Description**

Run alignment for a single sample

**Usage**

```r
run.alignment.sample(fastq.files, sample.id, output.directory = NULL, output.filename = NULL, code.directory = NULL, log.directory = NULL, config.file = NULL, job.dependencies = NULL, job.name = NULL, job.group = NULL, quiet = FALSE, verify.options = !quiet)
```

**Arguments**

- `fastq.files`  
  Paths to FASTQ files (one file if single-end reads, two files if paired-end)
- `sample.id`  
  Sample ID for labelling
- `output.directory`  
  Path to output directory
- `output.filename`  
  Name of resulting VCF file (defaults to SAMPLE_ID.vcf)
- `code.directory`  
  Path to directory where code should be stored
- `log.directory`  
  Path to directory where log files should be stored
- `config.file`  
  Path to config file
- `job.dependencies`  
  Vector with names of job dependencies
- `job.name`  
  Name of job to be submitted
- `job.group`  
  Group job should belong to
run.annotation

run.all.scripts

Description

Run all the scripts generated by previous parts of the pipeline, without using HPC commands

Usage

run.all.scripts(output.directory, stages.to.run = c("alignment", "qc", "calling", "annotation", "merging"), variant.callers = NULL, quiet = FALSE)

Arguments

output.directory
  Main directory where all files should be saved

stages.to.run
  A character vector of all stages that need running

variant.callers
  A character vector of variant callers to run

quiet
  Logical indicating whether to print commands to screen rather than submit jobs. Defaults to FALSE, can be useful to set to TRUE for testing.

Value

None

run.annotation

Description

Run annotation on a set of VCF files

Usage

run.annotation(vcf.specification, output.directory = NULL, job.name.prefix = NULL, job.group = NULL, quiet = FALSE, verify.options = !quiet)
**Arguments**

vcf.specification  
Data frame detailing VCF files to be processed, from prepare.vcf.specification.

output.directory  
Path to folder where code and log files should be stored in their respective sub-directories. If not supplied, code and log files will be stored in the directory with each VCF file.

job.name.prefix  
Prefix to be added before VCF name in job name. Defaults to 'annotate', but should be changed if running multiple callers to avoid

job.group  
Group job should be associated with on cluster

quiet  
Logical indicating whether to print commands to screen rather than submit them

verify.options  
Logical indicating whether to run verify.varitas.options

**Value**

Data frame with details of variant files

**Examples**

```r
run.annotation(
  data.frame(
    sample.id = c('a', 'b'),
    vcf = c('a.vcf', 'b.vcf'),
    caller = c('mutect', 'mutect')
  ),
  output.directory = '.',
  quiet = TRUE
)
```

---

**run.annovar.vcf**  
*Run ANNOVAR on a VCF file*

**Description**

Run ANNOVAR on a VCF file

**Usage**

```r
run.annovar.vcf(vcf.file, output.directory = NULL,
                 output.filename = NULL, code.directory = NULL,
                 log.directory = NULL, config.file = NULL, job.dependencies = NULL,
                 job.group = NULL, job.name = NULL, isis = FALSE, quiet = FALSE,
                 verify.options = !quiet)
```
Arguments

- `vcf.file` (Path to VCF file)
- `output.directory` (Path to output directory)
- `output.filename` (Name of resulting VCF file, defaults to SAMPLE_ID.vcf)
- `code.directory` (Path to directory where code should be stored)
- `log.directory` (Path to directory where log files should be stored)
- `config.file` (Path to config file)
- `job.dependencies` (Vector with names of job dependencies)
- `job.group` (Group job should belong to)
- `job.name` (Name of job to be submitted)
- `isis` (Logical indicating whether VCF files are from the isis (MiniSeq) variant caller)
- `quiet` (Logical indicating whether to print command to screen rather than submit it to the system. Defaults to false, useful for debugging.)
- `verify.options` (Logical indicating whether to run verify.varitas.options)

Value

- None

---

**run.filtering.txt**  
*Run filtering on an ANNOVAR-annotated txt file*

Description

Run filtering on an ANNOVAR-annotated txt file

Usage

```r
run.filtering.txt(variant.file, caller = c("consensus", "vardict", "ides", "mutect"), output.directory = NULL, output.filename = NULL, code.directory = NULL, log.directory = NULL, config.file = NULL, job.dependencies = NULL, job.group = NULL, quiet = FALSE)
```

Arguments

- `variant.file` (Path to variant file)
- `caller` (String giving variant caller that was used (affects which filters were applied.)
- `output.directory` (Path to output directory)
output.filename
   Name of resulting VCF file (defaults to SAMPLE_ID.vcf)

code.directory
   Path to directory where code should be stored

log.directory
   Path to directory where log files should be stored

config.file
   Path to config file

job.dependencies
   Vector with names of job dependencies

job.group
   Group job should belong to

quiet
   Logical indicating whether to print command to screen rather than submit it to
   the system. Defaults to false, useful for debugging.

---

**run.ides**

**Run iDES**

**Description**

Run iDES

**Usage**

```r
run.ides(project.directory, sample.id.pattern = "._S\d+\$", sample.ids = NULL, job.dependencies = NULL)
```

**Arguments**

- `project.directory`
  - Directory containing files
- `sample.id.pattern`
  - Regex pattern to match sample IDs
- `sample.ids`
  - Vector of sample IDs
- `job.dependencies`
  - Vector of job dependencies

**Details**

Run iDES step 1 on each sample, to tally up calls by strand. Files are output to a the sample subdirectory

**Value**

None

**Note**

Deprecated function for running iDES. Follows previous development package without specifica-
tion data frames
run.lofreq.sample

References

https://cappseq.stanford.edu/ides/

run.lofreq.sample  Run LoFreq for a sample

Description

Run LoFreq for a sample

Usage

run.lofreq.sample(tumour.bam, sample.id, paired, normal.bam = NULL,
output.directory = NULL, output.filename = NULL,
code.directory = NULL, log.directory = NULL, config.file = NULL,
job.dependencies = NULL, quiet = FALSE, job.name = NULL,
verify.options = !quiet, job.group = NULL)

Arguments

tumour.bam  Path to tumour sample BAM file.
sample.id  Sample ID for labelling
paired  Logical indicating whether to do variant calling with a matched normal.
normal.bam  Path to normal BAM file if paired = TRUE
output.directory  Path to output directory
output.filename  Name of resulting VCF file (defaults to SAMPLE_ID.vcf)
code.directory  Path to directory where code should be stored
log.directory  Path to directory where log files should be stored
config.file  Path to config file
job.dependencies  Vector with names of job dependencies
quiet  Logical indicating whether to print command to screen rather than submit it to
the system. Defaults to false, useful for debugging.
job.name  Name of job to be submitted
verify.options  Logical indicating whether to run verify.varitas.options
job.group  Group job should belong to
run.muse.sample  Run MuSE for a sample

Description

Run MuSE for a sample

Usage

run.muse.sample(tumour.bam, sample.id, paired, normal.bam = NULL,
output.directory = NULL, output.filename = NULL,
code.directory = NULL, log.directory = NULL, config.file = NULL,
job.dependencies = NULL, quiet = FALSE, job.name = NULL,
verify.options = !quiet, job.group = NULL)

Arguments

tumour.bam  Path to tumour sample BAM file.
sample.id  Sample ID for labelling
paired  Logical indicating whether to do variant calling with a matched normal.
normal.bam  Path to normal BAM file if paired = TRUE
output.directory  Path to output directory
output.filename  Name of resulting VCF file (defaults to SAMPLE_ID.vcf)
code.directory  Path to directory where code should be stored
log.directory  Path to directory where log files should be stored
config.file  Path to config file
job.dependencies  Vector with names of job dependencies
quiet  Logical indicating whether to print command to screen rather than submit it to the system. Defaults to false, useful for debugging.
job.name  Name of job to be submitted
verify.options  Logical indicating whether to run verify.varitas.options
job.group  Group job should belong to
run.mutect.sample

Run MuTect for a sample

Description

Run MuTect for a sample

Usage

```r
run.mutect.sample(tumour.bam, sample.id, paired, normal.bam = NULL,
output.directory = NULL, output.filename = NULL,
code.directory = NULL, log.directory = NULL, config.file = NULL,
job.dependencies = NULL, quiet = FALSE, job.name = NULL,
verify.options = !quiet, job.group = NULL)
```

Arguments

tumour.bam    Path to tumour sample BAM file.
sample.id     Sample ID for labelling
paired        Logical indicating whether to do variant calling with a matched normal.
normal.bam   Path to normal BAM file if paired = TRUE
output.directory Path to output directory
output.filename Name of resulting VCF file (defaults to SAMPLE_ID.vcf)
code.directory Path to directory where code should be stored
log.directory Path to directory where log files should be stored
config.file   Path to config file
job.dependencies    Vector with names of job dependencies
quiet    Logical indicating whether to print command to screen rather than submit it to the system. Defaults to false, useful for debugging.
job.name   Name of job to be submitted
verify.options Logical indicating whether to run verify.varitas.options
job.group   Group job should belong to
## run.post.processing

**Description**

Submit post-processing job to the cluster with appropriate job dependencies

**Usage**

```r
code.directory = NULL, log.directory = NULL, config.file = NULL,
job.name.prefix = NULL, quiet = FALSE, email = NULL,
verify.options = !quiet)
```

**Arguments**

- `variant.specification` Data frame specifying files to be processed
- `output.directory` Path to directory where output should be saved
- `code.directory` Directory where code should be saved
- `log.directory` Directory where log files should be saved
- `config.file` Path to config file
- `job.name.prefix` Prefix for job names on the cluster
- `quiet` Logical indicating whether to print commands to screen rather than submit the job
- `email` Email address that should be notified when job finishes. If NULL or FALSE, no email is sent
- `verify.options` Logical indicating whether `verify.varitas.options()` should be run.

**Value**

None

**Examples**

```r
run.post.processing(
  variant.specification = data.frame(
    sample.id = c('a', 'b'),
    vcf = c('a.vcf', 'b.vcf'),
    caller = c('mutect', 'mutect'),
    job.dependency = c('example1', 'example2')
  ),
  output.directory = '.',
  quiet = TRUE
)
```
run.target.qc

Perform sample QC by looking at target coverage.

Description

Perform sample QC by looking at target coverage.

Usage

run.target.qc(bam.specification, project.directory,
              sample.directories = TRUE, paired = FALSE,
              output.subdirectory = FALSE, quiet = FALSE, job.name.prefix = NULL,
              verify.options = FALSE, job.group = "target_qc")

Arguments

bam.specification
    Data frame containing details of BAM files to be processed, typically from
    prepare.bam.specification.

project.directory
    Path to project directory where code and log files should be saved

sample.directories
    Logical indicating whether output for each sample should be put in its own di-
    rectory (within output.directory)

paired
    Logical indicating whether the analysis is paired. This does not affect QC di-
    rectly, but means normal samples get nested

output.subdirectory
    If further nesting is required, name of subdirectory. If no further nesting, set to
    FALSE

quiet
    Logical indicating whether to print commands to screen rather than submit the
    job

job.name.prefix
    Prefix for job names on the cluster

verify.options
    Logical indicating whether to run verify.varitas.options

job.group
    Group job should be associated with on cluster
**run.target.qc.sample**  
*Get ontarget reads and run coverage quality control*

**Description**

Get ontarget reads and run coverage quality control

**Usage**

```r
run.target.qc.sample(bam.file, sample.id, output.directory = NULL, 
  code.directory = NULL, log.directory = NULL, config.file = NULL, 
  job.dependencies = NULL, job.name = NULL, job.group = NULL, 
  quiet = FALSE)
```

**Arguments**

- **bam.file**: Path to BAM file
- **sample.id**: Sample ID for labelling
- **output.directory**: Path to output directory
- **code.directory**: Path to directory where code should be stored
- **log.directory**: Path to directory where log files should be stored
- **config.file**: Path to config file
- **job.dependencies**: Vector with names of job dependencies
- **job.name**: Name of job to be submitted
- **job.group**: Group job should belong to
- **quiet**: Logical indicating whether to print command to screen rather than submit it to the system. Defaults to false, useful for debugging.

---

**run.vardict.sample**  
*run.vardict.sample*

**Description**

Run VarDict on a sample. Idea: have a low-level function that simply submits job to Perl, after BAM paths have been found. and output paths already have been decided upon

**Usage**

```r
run.vardict.sample(tumour.bam, sample.id, paired, proton = FALSE, 
  normal.bam = NULL, output.directory = NULL, output.filename = NULL, 
  code.directory = NULL, log.directory = NULL, config.file = NULL, 
  job.dependencies = NULL, job.name = NULL, job.group = NULL, 
  quiet = FALSE, verify.options = !quiet)
```
Arguments

tumour.bam: Path to tumour sample BAM file.
sample.id: Sample ID for labelling
paired: Logical indicating whether to do variant calling with a matched normal.
proton: Logical indicating whether the data was generated by proton sequencing. Defaults to False (i.e. Illumina)
normal.bam: Path to normal BAM file if paired = TRUE
output.directory: Path to output directory
output.filename: Name of resulting VCF file (defaults to SAMPLE_ID.vcf)
code.directory: Path to directory where code should be stored
log.directory: Path to directory where log files should be stored
config.file: Path to config file
job.dependencies: Vector with names of job dependencies
job.name: Name of job to be submitted
job.group: Group job should belong to
quiet: Logical indicating whether to print command to screen rather than submit it to the system. Defaults to false, useful for debugging.
verify.options: Logical indicating whether to run verify.varitas.options

Description

Run variant calling for all samples

Usage

run.variant.calling(bam.specification, output.directory,
variant.callers = c("vardict", "mutect", "varscan", "lofreq", "muse"),
paired = TRUE, proton = FALSE, sampledirectories = TRUE,
job.name.prefix = NULL, quiet = FALSE, verify.options = !quiet)
run.variant.calling

Arguments

bam.specification
   Data frame containing details of BAM files to be processed, typically from prepare.bam.specification.

output.directory
   Path to directory where output should be saved

variant.callers
   Character vector of variant callers to be used

paired
   Logical indicating whether to do variant calling with a matched normal

proton
   Logical indicating whether data was generated by proton sequencing (ignored if running MuTect)

sample.directories
   Logical indicating whether output for each sample should be put in its own directory (within output.directory)

job.name.prefix
   Prefix for job names on the cluster

quiet
   Logical indicating whether to print commands to screen rather than submit the job

verify.options
   Logical indicating whether to run verify.varitas.options

Details

Run VarDict on each sample, and annotate the results with ANNOVAR. Files are output to a vardict/subdirectory within each sample directory.

Value

None

Examples

run.variant.calling(
   data.frame(sample.id = c('Z', 'Y'), tumour.bam = c('Z.bam', 'Y.bam')),
   output.directory = '.',
   variant.caller = c('lofreq', 'mutect'),
   quiet = TRUE,
   paired = FALSE
)
Run VariTAS pipeline in full.

Description
Run all steps in VariTAS processing pipeline, with appropriate dependencies.

Usage
run.varitas.pipeline(file.details, output.directory, run.name = NULL, 
start.stage = c("alignment", "qc", "calling", "annotation", "merging"), 
variant.callers = NULL, proton = FALSE, quiet = FALSE, 
email = NULL, verify.options = !quiet, 
save.specification.files = !quiet)

Arguments
file.details Data frame containing details of files to be used during first processing step. Depending on what you want to be the first step in the pipeline, this can either be FASTQ files, BAM files, VCF files, or variant (txt) files.
output.directory Main directory where all files should be saved
run.name Name of pipeline run. Will be added as a prefix to all LSF jobs.
start.stage String indicating which stage pipeline should start at. If starting at a later stage of the pipeline, appropriate input files must be provided. For example, if starting with annotation, VCF files with variant calls must be provided.
variant.callers Vector specifying which variant callers should be run.
proton Logical indicating if data was generated by proton sequencing. Used to set base quality thresholds in variant calling steps.
quiet Logical indicating whether to print commands to screen rather than submit jobs. Defaults to FALSE, can be useful to set to TRUE for testing.
email Email address that should be notified when pipeline finishes. If NULL or FALSE, no email is sent.
verify.options Logical indicating whether to run verify.varitas.options
save.specification.files Logical indicating if specification files should be saved to project directory

Value
None
run.varitas.pipeline.hybrid

Examples

```r
run.varitas.pipeline(
    file.details = data.frame(
        sample.id = c('1', '2'),
        reads = c('1-R1.fastq.gz', '2-R1.fastq.gz'),
        mates = c('1-R2.fastq.gz', '2-R2.fastq.gz'),
        patient.id = c('P1', 'P1'),
        tissue = c('tumour', 'normal')
    ),
    output.directory = '.',
    quiet = TRUE,
    run.name = "Test",
    variant.callers = c('mutect', 'varscan')
)
```

Description

Run VariTAS pipeline starting from both VCF files and BAM/FASTQ files. Useful for processing data from the Ion PGM or MiniSeq where variant calling has been done on the machine, but you are interested in running more variant callers.

Usage

```r
run.varitas.pipeline.hybrid(vcf.specification, output.directory,
    run.name = NULL, fastq.specification = NULL,
    bam.specification = NULL, variant.callers = c("mutect", "vardict",
    "varscan", "lofreq", "muse"), proton = FALSE, quiet = FALSE,
    email = NULL, verify.options = !quiet,
    save.specification.files = !quiet)
```

Arguments

- **vcf.specification**
  - Data frame containing details of vcf files to be processed. Must contain columns sample.id, vcf, and caller
- **output.directory**
  - Main directory where all files should be saved
- **run.name**
  - Name of pipeline run. Will be added as a prefix to all LSF jobs.
- **fastq.specification**
  - Data frame containing details of FASTQ files to be processed
- **bam.specification**
  - Data frame containing details of BAM files to be processed
variant.callers  Vector specifying which variant callers should be run.
proton        Logical indicating if data was generated by proton sequencing. Used to set base
              quality thresholds in variant calling steps.
quiet         Logical indicating whether to print commands to screen rather than submit jobs.
              Defaults to FALSE, can be useful to set to TRUE for testing.
email         Email address that should be notified when pipeline finishes. If NULL or FALSE,
              no email is sent.
verify.options Logical indicating whether to run verify.varitas.options
save.specification.files Logical indicating if specification files should be saved to project directory

Value
None

Examples

run.varitas.pipeline.hybrid(
  bam.specification = data.frame(sample.id = c('Z', 'Y'), tumour.bam = c('Z.bam', 'Y.bam')),
  vcf.specification = data.frame(
    sample.id = c('a', 'b'),
    vcf = c('a.vcf', 'b.vcf'),
    caller = c('pgm', 'pgm')
  ),
  output.directory = '.',
  quiet = TRUE,
  run.name = "Test",
  variant.callers = c('mutect', 'varscan')
)

---

run.varscan.sample  Run VarScan for a sample

Description
Run VarScan for a sample

Usage

run.varscan.sample(tumour.bam, sample.id, paired, normal.bam = NULL,
  output.directory = NULL, output.filename = NULL,
  code.directory = NULL, log.directory = NULL, config.file = NULL,
  job.dependencies = NULL, quiet = FALSE, job.name = NULL,
  verify.options = !quiet, job.group = NULL)
save.config

Arguments

- **tumour.bam**: Path to tumour sample BAM file.
- **sample.id**: Sample ID for labelling.
- **paired**: Logical indicating whether to do variant calling with a matched normal.
- **normal.bam**: Path to normal BAM file if `paired = TRUE`.
- **output.directory**: Path to output directory.
- **output.filename**: Name of resulting VCF file (defaults to `SAMPLE_ID.vcf`).
- **code.directory**: Path to directory where code should be stored.
- **log.directory**: Path to directory where log files should be stored.
- **config.file**: Path to config file.
- **job.dependencies**: Vector with names of job dependencies.
- **quiet**: Logical indicating whether to print command to screen rather than submit it to the system. Defaults to false, useful for debugging.
- **job.name**: Name of job to be submitted.
- **verify.options**: Logical indicating whether to run `verify.varitas.options`.
- **job.group**: Group job should belong to.

Description

Save current varitas config options to a temporary file, and return filename.

Usage

```r
save.config(output.file = NULL)
```

Arguments

- **output.file**: Path to output file. If NULL (default), the config file will be saved as a temporary file.

Value

Path to config file
save.coverage.excel  
Save coverage statistics to multi-worksheet Excel file.

Description
Save coverage statistics to multi-worksheet Excel file.

Usage
save.coverage.excel(project.directory, file.name, overwrite = TRUE)

Arguments
- project.directory  
  Path to project directory
- file.name  
  Name of output file
- overwrite  
  Logical indicating whether to overwrite existing file if it exists.

Value
None

---

save.variants.excel  
Save variants to Excel.

Description
Makes an Excel workbook with variant calls. If filters are provided, these will be saved to an additional worksheet within the same file.

Usage
save.variants.excel(variants, file.name, filters = NULL, overwrite = TRUE)

Arguments
- variants  
  Data frame containing variants
- file.name  
  Name of output file
- filters  
  Optional list of filters to be saved
- overwrite  
  Logical indicating whether to overwrite exiting file if it exists. Defaults to TRUE for consistency with other R functions.
set.varitas.options  Set options for varitas pipeline.

Description
Set or overwrite options for the VariTAS pipeline. Nested options should be separated by a dot. For example, to update the reference genome for grch38, use reference_genome.grch38

Usage
set.varitas.options(...)

Arguments
...
   options to set

Value
None

Examples
## Not run:
set.varitas.options(reference_build = 'grch38');
set.varitas.options(
   filters.mutect.min_normal_depth = 10,
   filters.vardict.min_normal_depth = 10
);

## End(Not run)

split.on.column  split.on.column

Description
Split data frame on a concatenated column.

Usage
## S3 method for class 'on.column'
split(dat, column, split.character)
system.ls

Arguments

- **dat** Data frame to be processed
- **column** Name of column to split on
- **split.character** Pattern giving character to split column on

Value

Data frame after splitting on column

---

sum.dp4

Description

Simply calculates the depth of coverage of the variant allele given a string of DP4 values

Usage

```r
## S3 method for class 'dp4'
sum(dp4.str)
```

Arguments

- **dp4.str** String of DP4 values in the form "1234,1234,1234,1234"

---

system.ls

Run ls command

Description

Runs ls command on system. This is a workaround since list.files can not match patterns based on subdirectory structure.

Usage

```r
system.ls(pattern = "", directory = "", error = FALSE)
```

Arguments

- **pattern** pattern to match files
- **directory** base directory command should be run from
- **error** logical indicating whether to throw an error if no matching founds found. Defaults to False.
**tabular.mean**

**Value**

paths returned by `ls` command

---

**Description**

Calculate the mean of data in tabular format

**Usage**

`tabular.mean(values, frequencies, ...)`

**Arguments**

- `values` vector of values
- `frequencies` frequency corresponding to each value
- `...` Additional parameters passed to `sum`

**Value**

calculated mean

---

**tabular.median**

**Description**

Calculate the median of data in tabular format

**Usage**

`tabular.median(values, frequencies, ...)`

**Arguments**

- `values` Vector of values
- `frequencies` Frequency corresponding to each value
- `...` Additional parameters passed to `sum`

**Value**

calculated median
trinucleotide.barplot  
*Make barplot of trinucleotide substitutions*

**Description**

Make barplot of trinucleotide substitutions

**Usage**

```
trinucleotide.barplot(variants, file.name)
```

**Arguments**

- `variants`  
  Data frame with variants
- `file.name`  
  Name of output file

**Value**

None

---

variant.recurrence.barplot  
*Make barplot of variants per caller*

**Description**

Make barplot of variants per caller

**Usage**

```
variant.recurrence.barplot(variants, file.name)
```

**Arguments**

- `variants`  
  Data frame with variants
- `file.name`  
  Name of output file

**Value**

None
variants.caller.barplot

Make barplot of variants per caller

Description

Make barplot of variants per caller

Usage

variants.caller.barplot(variants, file.name, group.by = NULL)

Arguments

variants Data frame with variants
file.name Name of output file
group.by Optional grouping variable for barplot

Value

None

variants.sample.barplot

Make barplot of variants per sample

Description

Make barplot of variants per sample

Usage

variants.sample.barplot(variants, file.name)

Arguments

variants Data frame with variants
file.name Name of output file

Value

None
verify.bam.specification

Check that sample specification data frame matches expected format, and that all files exist

Description
Check that sample specification data frame matches expected format, and that all files exist

Usage
verify.bam.specification(bam.specification)

Arguments
bam.specification
Data frame containing columns sample.id and tumour.bam, and optionally a column normal.bam.

Value
None

verify.bwa.index

Verify that bwa index files exist for a fasta file

Description
Verify that bwa index files exist for a fasta file

Usage
verify.bwa.index(fasta.file, error = FALSE)

Arguments
fasta.file
Fasta file to check

error
Logical indicating whether to throw an (informative) error if verification fails

Value
index.files.exist Logical indicating if bwa index files were found (only returned if error set to FALSE)
verify.fasta.index

Description
Verify that fasta index files exist for a given fasta file.

Usage
verify.fasta.index(fasta.file, error = FALSE)

Arguments
- fasta.file: Fasta file to check
- error: Logical indicating whether to throw an (informative) error if verification fails

Value
faidx.exists Logical indicating if fasta index files were found (only returned if error set to FALSE)

verify.fastq.specification

Check that FASTQ specification data frame matches expected format, and that all files exist

Description
Check that FASTQ specification data frame matches expected format, and that all files exist

Usage
verify.fastq.specification(fastq.specification, paired.end = FALSE, files.ready = FALSE)

Arguments
- fastq.specification: Data frame containing columns sample.id and reads, and optionally a column mates
- paired.end: Logical indicating whether paired end reads are used
- files.ready: Logical indicating if the files already exist on disk. If there are job dependencies, this should be set to FALSE.

Value
None
**verify.sequence.dictionary**

Description

Verify that sequence dictionary exists for a fasta file.

Usage

```
verify.sequence.dictionary(fasta.file, error = FALSE)
```

Arguments

- `fasta.file`: Fasta file to check
- `error`: Logical indicating whether to throw an (informative) error if verification fails

Value

`dict.exists` Logical indicating if sequence dictionary files were found (only returned if error set to FALSE)

**verify.varitas.options**

*Check against common errors in the VariTAS options.*

Description

Check against common errors in the VariTAS options before launching into pipeline.

Usage

```
verify.varitas.options(stages.to.run = c("alignment", "qc", "calling", 
  "annotation", "merging"), variant.callers = c("mutect", "vardict", 
  "ides", "varsan", "lofreq", "muse"), varitas.options = NULL)
```

Arguments

- `stages.to.run`: Vector indicating which stages should be run. Defaults to all possible stages. If only running a subset of stages, only checks corresponding to the desired stages are run
- `variant.callers`: Vector indicating which variant callers to run. Only used if calling is in `stages.to.run`
- `varitas.options`: Optional file path or list of VariTAS options.
Value

None

Description

Verify that VCF specification data frame fits expected format

Usage

`verify.vcf.specification(vcf.specification)`

Arguments

- `vcf.specification`
  
  VCF specification data frame

Value

None
Index

add.option, 4
alternate.gene.sort, 4
build.variant.specification, 5
caller.overlap.venn.diagram, 5
capitalise.caller (capitalize.caller), 6
capitalize.caller, 6
classify.variant, 6
convert.ides.output, 7
create.directories, 7
date.stamp.file.name, 8
datestamp.file.name
  (date.stamp.file.name), 8
datestamp.filename
  (date.stamp.file.name), 8
extract.sample.ids, 8
filter.variant.file, 9
filter.variants, 9
fix.lofreq.af, 10
fix.names, 10
fix.varscan.af, 11
get.base.substitution, 11
get.bed.chromosomes, 12
get.buildver, 12
get.colours, 13
get.coverage.by.amplicon, 13
get.coverage.by.sample.statistics, 14
get.fasta.chromosomes, 14
get.file.path, 15
get.filters, 15
gene, 16
get.miniseq.sample.files, 16
get.option, 17
get.pool.from.panel.data, 18
get.varitas.options, 18
get.vcf.chromosomes, 19
in.varitas.options, 19
logical.to.character, 20
make.command.line.call, 20
mean.field.value, 21
merge.ides.annotation, 21
merge.variants, 22
overwrite.varitas.options, 23
parse.job.dependencies, 23
plot.amplicon.coverage.per.sample, 24
plot.coverage.by.genome.order, 24
plot.coverage.by.sample, 25
plot.ontarget.percent, 25
plot.paired.percent, 26
post.processing, 26
prepare.bam.specification, 27
prepare.fastq.specification, 28
prepare.miniseq.specifications, 29
prepare.vcf.specification, 30
process.coverage.reports, 30
process.sample.contamination.checks, 31
process.total.coverage.statistics, 31
read.all.calls, 32
read.ides.file, 32
read.variant.calls, 33
read.yaml, 33
run.alignment, 34
run.alignment.sample, 35
run.all.scripts, 36
run.annotation, 36
run.annovar.vcf, 37
run.filtering.txt, 38
run.ides, 39
run.lofreq.sample, 40