# An Introduction to VariantTools

Michael Lawrence, Jeremiah Degenhardt

October 2, 2012

## Contents

1	Introduction	<b>2</b>
<b>2</b>	Calling single-sample variants	<b>2</b>
	2.1 Basic usage	2
	2.2 Step by step	3
	2.3 Diagnosing the filters	3
	2.4 Extending and customizing the workflow	7
3	Exporting the calls as VCF	7

## 1 Introduction

This vignette outlines the basic usages of the *VariantTools* package and the general workflow for loading data, calling single sample variants and tumor-specific somatic mutations or other sample-specific variant types (eg RNA editing). Most of the functions operate on alignments (BAM files) or datasets of called variants. The user is expected to have already aligned the reads with a separate tool, e.g., GSNAP via *gmapR*.

## 2 Calling single-sample variants

### 2.1 Basic usage

For our example, we take paired-end RNA-seq alignments from two lung cancer cell lines from the same individual. H1993 is derived from a metastatis and H2073 is derived from the primary tumor.

Below, we call variants from a region around the p53 gene:

```
> library(VariantTools)
> bams <- LungCancerLines::LungCancerBamFiles()
> bam <- bams$H1993
> tally.param <- VariantTallyParam(gmapR::TP53Genome(),
+ readlen = 100L,
+ high_base_quality = 23L,
+ which = range(p53))
> called.variants <- callVariants(bam, tally.param)</pre>
```

In the above, we load the genome corresponding to the human p53 gene region and the H1993 BAM file (stripped down to the same region). We pass the BAM, genome, read length and quality cutoff to the callVariants workhorse. The read length is not strictly required, but it is necessary for one of the QA filters. The value given for the high base quality cutoff is appropriate for Sanger and Illumina 1.8 or above. By default, the high quality counts are used by the likelihood ratio test during calling.

The returned called\_variants is a variant *GRanges*, in the same form as that returned by bam\_tally in the *gmapR* package. Unsurprisingly, callVariants uses bam\_tally internally to generate the per-nucleotide counts (pileup) from the BAM file. The result is then filtered to generate the variant calls. The *VCF* class holds similar information; however, we favor the simple tally *GRanges*, because it has a separate record for each ALT, at each position. *VCF*, the class and the file format, has a single record for a position, collapsing over multiple ALT alleles, and this is much less convenient for our purposes.

If we subset the variants by those in an actual p53 exon (not an intron), we find two: one with strong evidence for a homozygous mutation, and another with much weaker evidence (low coverage).

```
> subsetByOverlaps(called.variants, p53, ignore.strand = TRUE)
```

GRanges with 2 ranges and 20 metadata columns:

	seqnames	:	ranges	strand	Ι	100	cation		ref	
	<rle></rle>	<ir< td=""><td>anges&gt;</td><td><rle></rle></td><td>Ι</td><td><chara< td=""><td>acter&gt;</td><td><cha< td=""><td>aracter&gt;</td><td></td></cha<></td></chara<></td></ir<>	anges>	<rle></rle>	Ι	<chara< td=""><td>acter&gt;</td><td><cha< td=""><td>aracter&gt;</td><td></td></cha<></td></chara<>	acter>	<cha< td=""><td>aracter&gt;</td><td></td></cha<>	aracter>	
[1]	TP53 [10	012027, 10	12027]	+	Ι	TP53:10	012027		Т	
[2]	TP53 [10	013309, 10	13309]	+	Ι	TP53:10	013309		C	
	alt	ncycles	ncycle	es.ref		$\mathtt{count}$	count	.ref	count.to	tal
	<character></character>	<integer></integer>	<int< td=""><td>teger&gt; ·</td><td><i< td=""><td>nteger&gt;</td><td><integ< td=""><td>ger&gt;</td><td><integ< td=""><td>ger&gt;</td></integ<></td></integ<></td></i<></td></int<>	teger> ·	<i< td=""><td>nteger&gt;</td><td><integ< td=""><td>ger&gt;</td><td><integ< td=""><td>ger&gt;</td></integ<></td></integ<></td></i<>	nteger>	<integ< td=""><td>ger&gt;</td><td><integ< td=""><td>ger&gt;</td></integ<></td></integ<>	ger>	<integ< td=""><td>ger&gt;</td></integ<>	ger>
[1]	C	2		0		2		0		2
[2]	G	126		0		934		0		936
	high.quality	y high.qua	lity.re	ef high	.qı	uality.	total r	nean	.quality	
	<integer></integer>	> <	integei	r>		<inte< td=""><td>eger&gt;</td><td><r< td=""><td>numeric&gt;</td><td></td></r<></td></inte<>	eger>	<r< td=""><td>numeric&gt;</td><td></td></r<>	numeric>	
[1]		2		0			2	3	39.50000	

[2] 889 0 889 36.26884 mean.quality.ref count.pos count.pos.ref count.neg count.neg.ref <integer> <integer> <numeric> <integer> <integer> [1] <NA> 0 0 1 1 [2] <NA> 409 0 525 0 cycleCount.0.10 cycleCount.10.90 cycleCount.90.100 <integer> <integer> <integer> [1] 0 2 0 [2] 800 76 58 seqlengths: TP53 2025767

The next section goes into further detail on the process, including the specific filtering rules applied, and how one might, for example, tweak the parameters to avoid calling low-coverage variants, like the one above.

#### 2.2 Step by step

The callVariants method for BAM files, introduced above, is a convenience wrapper that delegates to several low-level functions to perform each step of the variant calling process: generating the tallies, basic QA filtering and the actual variant calling. Calling these functions directly affords the user more control over the process and provides access to intermediate results, which is useful e.g. for diagnostics and for caching results. The workflow consists of three function calls that rely on argument defaults to achieve the same result as our call to callVariants above. Please see their man pages for the arguments available for customization.

The first step is to tally the variants from the BAM file. By default, this will return observed differences from the reference, excluding N calls and only counting reads above 13 in mapping quality (MAPQ) score. There are three cycle bins: the first 10 bases, the final 10 bases, and the stretch between them (these will be used in the QA step).

#### > raw.variants <- tallyVariants(bam, tally.param)</pre>

Next, basic QA filters are applied. These include a minimum read count (2) check, minimum unique cycle count (2) check, and Fisher Exact Test on the per-strand counts vs. reference for strand bias (p-value cutoff: 0.001). If there are at least three cycle bins in the tallies, at least one read must present the variant in an internal cycle bin. The intent is to ensure that we have sufficient data and that the data are not due to strand-specific nor cycle-specific artifacts.

#### > qa.variants <- qaVariants(raw.variants)

The final step is to actually call the variants. The callVariants function uses a binomial likelihood ratio test for this purpose. The ratio is  $P(D|p = p_{lower})/P(D|p = p_{error})$ , where  $p_{lower} = 0.2$  is the assumed lowest variant frequency and  $p_{error} = 0.001$  is the assumed error rate in the sequencing (default: 0.001).

#### > called.variants <- callVariants(qa.variants)</pre>

#### 2.3 Diagnosing the filters

The calls to qaVariants and callVariants are essentially filtering the tallies, so it is important to know, especially when faced with a new dataset, the effect of each filter and the effect of the individual parameters on each filter.

The filters are implemented as modules and are stored in a *FilterRules* object from the *IRanges* package. We can create those filters directly and rely on some *FilterRules* utilities to diagnose the filtering process. Here we construct the *FilterRules* that implements the qaVariants function. Again, we rely on the argument defaults to generate the same answer.

```
> qa.filters <- VariantQAFilters()</pre>
```

We can now ask for a summary of the filtering process, which gives the number of variants that pass each filter, separately and then combined:

```
> summary(qa.filters, raw.variants)
```

<initial> nonNRef cycleCount fisherStrand cycleBin 3924 3924 1385 3852 3486 <final> 1281

Now we retrieve the variants that pass the filters:

```
> qa.variants <- subsetByFilter(raw.variants, qa.filters)</pre>
```

We could do the same, except modify a filter parameter, such as the p-value cutoff for the Fisher Exact Test for strand bias:

```
> qa.filters.custom <- VariantQAFilters(fisher.strand.p.value = 1e-4)
> summary(qa.filters.custom, raw.variants)
```

<initial></initial>	nonNRef	cycleCount	fisherStrand	cycleBin
3924	3924	1385	3876	3486
<final></final>				
1305				

To get a glance at the additional variants we are discarding compared to the previous cutoff, we can subset the filter sets down to the Fisher strand filter, evaluate the old and new filter, and compare the results:

```
> fs.original <- eval(qa.filters["fisherStrand"], raw.variants)
> fs.custom <- eval(qa.filters.custom["fisherStrand"], raw.variants)
> raw.variants[fs.original != fs.custom]
```

GRanges with 24 ranges and 20 metadata columns:

					0		0
ref	location	I	$\operatorname{strand}$	ranges		seqnames	
<character></character>	<character></character>	I	<rle></rle>	<iranges></iranges>		<rle></rle>	
Т	TP53:1010944	I	+	1010944]	[1010944,	TP53	[1]
C	TP53:1011428	- 1	+	1011428]	[1011428,	TP53	[2]
А	TP53:1011435	I	+	1011435]	[1011435,	TP53	[3]
Т	TP53:1011467	I	+	1011467]	[1011467,	TP53	[4]
Т	TP53:1012605	I	+	1012605]	[1012605,	TP53	[5]
Т	TP53:1013712	I	+	1013712]	[1013712,	TP53	[6]
Т	TP53:1013961	I	+	1013961]	[1013961,	TP53	[7]
Т	TP53:1017881	I	+	1017881]	[1017881,	TP53	[8]
Т	TP53:1017955	I	+	1017955]	[1017955,	TP53	[9]
Т	TP53:1018524	I	+	1018524]	[1018524,	TP53	[16]
Т	TP53:1018529	I	+	1018529]	[1018529,	TP53	[17]
G	TP53:1018669	I	+	1018669]	[1018669,	TP53	[18]
G	TP53:1018722	I	+	1018722]	[1018722,	TP53	[19]
G	TP53:1018738	I	+	1018738]	[1018738,	TP53	[20]

[21]	TP53 [101	8754, 1018754]	+	TP53:	:1018754	Т
[22]	TP53 [101	8807 1018807]	+	I TP53	1018807	т
[23]	TD53 [101	8843 1018843]	+	TD53	10188/3	- T
[20]	TDE2 [101	0062 1010062]			1010010	т т
[24]	1103 [101	0903, 1010903]	т f	1 1100.	.1010903	1
	alt	ncycles ncycles	rei	count	count.rei	
	<character> &lt;</character>	integer> <integ< td=""><td>ger&gt; <i< td=""><td>nteger&gt;</td><td><integer></integer></td><td></td></i<></td></integ<>	ger> <i< td=""><td>nteger&gt;</td><td><integer></integer></td><td></td></i<>	nteger>	<integer></integer>	
[1]	G	16	112	23	629	
[2]	G	7	140	8	634	
[3]	C	13	136	41	596	
[4]	C	6	142	12	744	
[5]	G	6	124	8	778	
[6]	G	7	139	7	905	
[7]	C	6	134	9	699	
[8]	С	5	90	6	385	
[9]	С	6	130	7	778	
[0]	· ·	C C	200			
 [16]	· · · ·		133	۰ م	1002	
	C	0	100	9	1002	
		4	120	14	909 710	
[10]	1	9	93	14	712	
[19]	A	4	89	4	605	
[20]	Т	3	90	4	687	
[21]	G	3	88	5	705	
[22]	C	8	93	18	417	
[23]	G	10	108	14	509	
[24]	G	9	117	11	563	
	count.total h	igh.quality high	qualit	y.ref hi	igh.quality	.total
	<pre>count.total h    <integer></integer></pre>	<pre>igh.quality high   <integer></integer></pre>	qualit. int	y.ref hi eger>	igh.quality <in< td=""><td>r.total nteger&gt;</td></in<>	r.total nteger>
[1]	count.total h <integer> 653</integer>	igh.quality high <integer> 0</integer>	qualit. <int< td=""><td>y.ref hi eger&gt; 416</td><td>igh.quality <in< td=""><td>r.total hteger&gt; 416</td></in<></td></int<>	y.ref hi eger> 416	igh.quality <in< td=""><td>r.total hteger&gt; 416</td></in<>	r.total hteger> 416
[1] [2]	count.total h <integer> 653 643</integer>	igh.quality high <integer> 0 0</integer>	qualit. <int< td=""><td>y.ref hi eger&gt; 416 493</td><td>igh.quality <in< td=""><td>r.total hteger&gt; 416 493</td></in<></td></int<>	y.ref hi eger> 416 493	igh.quality <in< td=""><td>r.total hteger&gt; 416 493</td></in<>	r.total hteger> 416 493
[1] [2] [3]	count.total h <integer> 653 643 638</integer>	igh.quality high <integer> 0 0 1</integer>	qualit. <int< td=""><td>y.ref hi eger&gt; 416 493 454</td><td>igh.quality <in< td=""><td>r.total uteger&gt; 416 493 455</td></in<></td></int<>	y.ref hi eger> 416 493 454	igh.quality <in< td=""><td>r.total uteger&gt; 416 493 455</td></in<>	r.total uteger> 416 493 455
[1] [2] [3] [4]	count.total h <integer> 653 643 638 758</integer>	<pre>igh.quality high <integer> 0 0 1 0</integer></pre>	.qualit <int< td=""><td>y.ref hi eger&gt; 416 493 454 621</td><td>igh.quality <in< td=""><td>r.total ateger&gt; 416 493 455 621</td></in<></td></int<>	y.ref hi eger> 416 493 454 621	igh.quality <in< td=""><td>r.total ateger&gt; 416 493 455 621</td></in<>	r.total ateger> 416 493 455 621
[1] [2] [3] [4]	count.total h <integer> 653 643 638 758 788</integer>	<pre>igh.quality high   <integer>     0     0     1     0     0     0 </integer></pre>	.qualit <int< td=""><td>y.ref hi eger&gt; 416 493 454 621 668</td><td>igh.quality <in< td=""><td>r.total ateger&gt; 416 493 455 621 668</td></in<></td></int<>	y.ref hi eger> 416 493 454 621 668	igh.quality <in< td=""><td>r.total ateger&gt; 416 493 455 621 668</td></in<>	r.total ateger> 416 493 455 621 668
[1] [2] [3] [4] [5]	count.total h <integer> 653 643 638 758 788 913</integer>	<pre>igh.quality high   <integer>     0     0     1     0</integer></pre>	.qualit <int< td=""><td>y.ref hi eger&gt; 416 493 454 621 668 836</td><td>igh.quality <in< td=""><td>r.total 416 493 455 621 668 836</td></in<></td></int<>	y.ref hi eger> 416 493 454 621 668 836	igh.quality <in< td=""><td>r.total 416 493 455 621 668 836</td></in<>	r.total 416 493 455 621 668 836
[1] [2] [3] [4] [5] [6]	count.total h <integer> 653 643 638 758 788 913 711</integer>	<pre>igh.quality high</pre>	.qualit <int< td=""><td>y.ref hi eger&gt; 416 493 454 621 668 836 593</td><td>igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593</td></in<></td></int<>	y.ref hi eger> 416 493 454 621 668 836 593	igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593</td></in<>	r.total 416 493 455 621 668 836 593
[1] [2] [3] [4] [6] [7]	count.total h <integer> 653 643 638 758 788 913 711 202</integer>	<pre>igh.quality high     <integer></integer></pre>	.qualit <int< td=""><td>y.ref hi eger&gt; 416 493 454 621 668 836 593 227</td><td>igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 227</td></in<></td></int<>	y.ref hi eger> 416 493 454 621 668 836 593 227	igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 227</td></in<>	r.total 416 493 455 621 668 836 593 227
[1] [2] [3] [4] [5] [6] [7] [8]	count.total h <integer> 653 643 638 758 788 913 711 393 702</integer>	<pre>igh.quality high     <integer></integer></pre>	.qualit <int< td=""><td>y.ref hi eger&gt; 416 493 454 621 668 836 593 327 712</td><td>igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712</td></in<></td></int<>	y.ref hi eger> 416 493 454 621 668 836 593 327 712	igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712</td></in<>	r.total 416 493 455 621 668 836 593 327 712
[1] [2] [3] [4] [5] [6] [7] [8] [9]	count.total h <integer> 653 643 638 758 788 913 711 393 792</integer>	<pre>igh.quality high     <integer></integer></pre>	.qualit <int< td=""><td>y.ref hi eger&gt; 416 493 454 621 668 836 593 327 712</td><td>igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712</td></in<></td></int<>	y.ref hi eger> 416 493 454 621 668 836 593 327 712	igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712</td></in<>	r.total 416 493 455 621 668 836 593 327 712
<ol> <li>[1]</li> <li>[2]</li> <li>[3]</li> <li>[4]</li> <li>[5]</li> <li>[6]</li> <li>[7]</li> <li>[8]</li> <li>[9]</li> <li></li> </ol>	count.total h <integer> 653 643 638 758 788 913 711 393 792 </integer>	<pre>igh.quality high     <integer></integer></pre>	.qualit <int< td=""><td>y.ref hi eger&gt; 416 493 454 621 668 836 593 327 712 </td><td>igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712 </td></in<></td></int<>	y.ref hi eger> 416 493 454 621 668 836 593 327 712 	igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712 </td></in<>	r.total 416 493 455 621 668 836 593 327 712 
[1] [2] [3] [4] [5] [6] [7] [8] [9] 	count.total h <integer> 653 643 638 758 788 913 711 393 792  1013</integer>	<pre>igh.quality high</pre>	.qualit <int< td=""><td>y.ref hi eger&gt; 416 493 454 621 668 836 593 327 712  842</td><td>igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842</td></in<></td></int<>	y.ref hi eger> 416 493 454 621 668 836 593 327 712  842	igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842</td></in<>	r.total 416 493 455 621 668 836 593 327 712  842
[1] [2] [3] [4] [5] [6] [7] [8] [9]  [16] [17]	count.total h <integer> 653 643 638 758 788 913 711 393 792  1013 972</integer>	<pre>igh.quality high     <integer></integer></pre>	.qualit <int< td=""><td>y.ref hi eger&gt; 416 493 454 621 668 836 593 327 712  842 828</td><td>igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842 828</td></in<></td></int<>	y.ref hi eger> 416 493 454 621 668 836 593 327 712  842 828	igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842 828</td></in<>	r.total 416 493 455 621 668 836 593 327 712  842 828
[1] [2] [3] [4] [5] [6] [7] [8] [9]  [16] [17] [18]	count.total h <integer> 653 643 638 758 788 913 711 393 792  1013 972 728</integer>	<pre>igh.quality high     <integer></integer></pre>	.qualit <int< td=""><td>y.ref hi eger&gt; 416 493 454 621 668 836 593 327 712  842 828 577</td><td>igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842 828 577</td></in<></td></int<>	y.ref hi eger> 416 493 454 621 668 836 593 327 712  842 828 577	igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842 828 577</td></in<>	r.total 416 493 455 621 668 836 593 327 712  842 828 577
[1] [2] [3] [4] [5] [6] [7] [8] [9]  [16] [17] [18] [19]	count.total h <integer> 653 643 638 758 788 913 711 393 792  1013 972 728 611</integer>	<pre>igh.quality high     <integer></integer></pre>	.qualit <int< td=""><td>y.ref hi eger&gt; 416 493 454 621 668 836 593 327 712  842 828 577 543</td><td>igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842 828 577 543</td></in<></td></int<>	y.ref hi eger> 416 493 454 621 668 836 593 327 712  842 828 577 543	igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842 828 577 543</td></in<>	r.total 416 493 455 621 668 836 593 327 712  842 828 577 543
[1] [2] [3] [4] [5] [6] [7] [8] [9]  [16] [17] [18] [19] [20]	count.total h <integer> 653 643 638 758 788 913 711 393 792  1013 972 728 611 691</integer>	<pre>igh.quality high     <integer></integer></pre>	.qualit <int< td=""><td>y.ref hi eger&gt; 416 493 454 621 668 836 593 327 712  842 828 577 543 657</td><td>igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842 828 577 543 657</td></in<></td></int<>	y.ref hi eger> 416 493 454 621 668 836 593 327 712  842 828 577 543 657	igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842 828 577 543 657</td></in<>	r.total 416 493 455 621 668 836 593 327 712  842 828 577 543 657
[1] [2] [3] [4] [5] [6] [7] [8] [9]  [16] [17] [18] [19] [20] [21]	count.total h <integer> 653 643 638 758 788 913 711 393 792  1013 972 728 611 691 713</integer>	<pre>igh.quality high     <integer></integer></pre>	.qualit <int< td=""><td>y.ref hi eger&gt; 416 493 454 621 668 836 593 327 712  842 828 577 543 657 662</td><td>igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662</td></in<></td></int<>	y.ref hi eger> 416 493 454 621 668 836 593 327 712  842 828 577 543 657 662	igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662</td></in<>	r.total 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662
[1] [2] [3] [4] [5] [6] [7] [8]  [16] [17] [18] [19] [20] [21] [22]	count.total h <integer> 653 643 638 758 788 913 711 393 792  1013 972 728 611 691 713 436</integer>	<pre>igh.quality high     <integer></integer></pre>	.qualit <int< td=""><td>y.ref hi eger&gt; 416 493 454 621 668 836 593 327 712  842 828 577 543 657 662 326</td><td>igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662 327</td></in<></td></int<>	y.ref hi eger> 416 493 454 621 668 836 593 327 712  842 828 577 543 657 662 326	igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662 327</td></in<>	r.total 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662 327
[1] [2] [3] [4] [5] [6] [7] [8] [9]  [16] [17] [18] [19] [20] [21] [22] [23]	count.total h <integer> 653 643 638 758 788 913 711 393 792  1013 972 728 611 691 713 436 525</integer>	<pre>igh.quality high     <integer></integer></pre>	.qualit <int< td=""><td>y.ref hi eger&gt; 416 493 454 621 668 836 593 327 712  842 828 577 543 657 662 326 437</td><td>igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662 327 437</td></in<></td></int<>	y.ref hi eger> 416 493 454 621 668 836 593 327 712  842 828 577 543 657 662 326 437	igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662 327 437</td></in<>	r.total 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662 327 437
<ul> <li>[1]</li> <li>[2]</li> <li>[3]</li> <li>[4]</li> <li>[5]</li> <li>[6]</li> <li>[7]</li> <li>[8]</li> <li>[9]</li> <li></li> <li>[16]</li> <li>[17]</li> <li>[18]</li> <li>[19]</li> <li>[20]</li> <li>[21]</li> <li>[22]</li> <li>[23]</li> <li>[24]</li> </ul>	count.total h <integer> 653 643 638 758 788 913 711 393 792  1013 972 728 611 691 713 436 525 577</integer>	<pre>igh.quality high     <integer></integer></pre>	.qualit <int< td=""><td>y.ref hi eger&gt; 416 493 454 621 668 836 593 327 712  842 828 577 543 657 662 326 437 473</td><td>igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662 327 437 473</td></in<></td></int<>	y.ref hi eger> 416 493 454 621 668 836 593 327 712  842 828 577 543 657 662 326 437 473	igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662 327 437 473</td></in<>	r.total 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662 327 437 473
<ul> <li>[1]</li> <li>[2]</li> <li>[3]</li> <li>[4]</li> <li>[5]</li> <li>[6]</li> <li>[7]</li> <li>[8]</li> <li>[9]</li> <li></li> <li>[16]</li> <li>[17]</li> <li>[18]</li> <li>[19]</li> <li>[20]</li> <li>[21]</li> <li>[22]</li> <li>[23]</li> <li>[24]</li> </ul>	count.total h <integer> 653 643 638 758 788 913 711 393 792  1013 972 728 611 691 713 436 525 577 mean.quality</integer>	<pre>igh.quality high     <integer></integer></pre>	.qualit <int< td=""><td><pre>y.ref hi eger&gt;     416     493     454     621     668     836     593     327     712      842     828     577     543     657     662     326     437     473 pos cour</pre></td><td>igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662 327 437 437 473 count.neg</td></in<></td></int<>	<pre>y.ref hi eger&gt;     416     493     454     621     668     836     593     327     712      842     828     577     543     657     662     326     437     473 pos cour</pre>	igh.quality <in< td=""><td>r.total 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662 327 437 437 473 count.neg</td></in<>	r.total 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662 327 437 437 473 count.neg
<ul> <li>[1]</li> <li>[2]</li> <li>[3]</li> <li>[4]</li> <li>[5]</li> <li>[6]</li> <li>[7]</li> <li>[8]</li> <li>[9]</li> <li></li> <li>[16]</li> <li>[17]</li> <li>[18]</li> <li>[19]</li> <li>[20]</li> <li>[21]</li> <li>[22]</li> <li>[23]</li> <li>[24]</li> </ul>	count.total h <integer> 653 643 638 758 788 913 711 393 792  1013 972 728 611 691 713 436 525 577 mean.quality <numeric></numeric></integer>	<pre>igh.quality high     <integer></integer></pre>	<pre>count. <integ< pre=""></integ<></pre>	<pre>y.ref hi eger&gt;     416     493     454     621     668     836     593     327     712      842     828     577     543     657     662     326     437     473 pos cour er&gt;</pre>	igh.quality <in <int.pos.ref <integer></integer></int.pos.ref </in 	r.total hteger> 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662 327 437 473 count.neg <integer></integer>
<ul> <li>[1]</li> <li>[2]</li> <li>[3]</li> <li>[4]</li> <li>[5]</li> <li>[6]</li> <li>[7]</li> <li>[8]</li> <li>[9]</li> <li></li> <li>[16]</li> <li>[17]</li> <li>[18]</li> <li>[20]</li> <li>[21]</li> <li>[22]</li> <li>[23]</li> <li>[24]</li> </ul>	count.total h <integer> 653 643 638 758 788 913 711 393 792  1013 972 728 611 691 713 436 525 577 mean.quality <numeric> <na></na></numeric></integer>	<pre>igh.quality high     <integer></integer></pre>	count. <integ< td=""><td><pre>y.ref hi eger&gt;     416     493     454     621     668     836     593     327     712      842     828     577     543     657     662     326     437     473 pos cour er&gt;     23</pre></td><td>igh.quality <in <integer> 444</integer></in </td><td>r.total hteger&gt; 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662 327 437 473 count.neg <integer> 0</integer></td></integ<>	<pre>y.ref hi eger&gt;     416     493     454     621     668     836     593     327     712      842     828     577     543     657     662     326     437     473 pos cour er&gt;     23</pre>	igh.quality <in <integer> 444</integer></in 	r.total hteger> 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662 327 437 473 count.neg <integer> 0</integer>
<ul> <li>[1]</li> <li>[2]</li> <li>[3]</li> <li>[4]</li> <li>[5]</li> <li>[6]</li> <li>[7]</li> <li>[8]</li> <li>[9]</li> <li></li> <li>[16]</li> <li>[17]</li> <li>[20]</li> <li>[21]</li> <li>[21]</li> <li>[21]</li> </ul>	count.total h <integer> 653 643 638 758 788 913 711 393 792  1013 972 728 611 691 713 436 525 577 mean.quality <numeric> <na></na></numeric></integer>	<pre>igh.quality high     <integer></integer></pre>	count. <integ< td=""><td><pre>y.ref hi eger&gt;     416     493     454     621     668     836     593     327     712      842     828     577     543     657     662     326     437     473 pos cour er&gt;     23     8</pre></td><td>nt.pos.ref <integer> 444 258</integer></td><td>r.total hteger&gt; 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662 327 437 473 count.neg <integer> 0 0</integer></td></integ<>	<pre>y.ref hi eger&gt;     416     493     454     621     668     836     593     327     712      842     828     577     543     657     662     326     437     473 pos cour er&gt;     23     8</pre>	nt.pos.ref <integer> 444 258</integer>	r.total hteger> 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662 327 437 473 count.neg <integer> 0 0</integer>
<ul> <li>[1]</li> <li>[2]</li> <li>[3]</li> <li>[4]</li> <li>[5]</li> <li>[6]</li> <li>[7]</li> <li>[8]</li> <li>[9]</li> <li></li> <li>[16]</li> <li>[17]</li> <li>[20]</li> <li>[21]</li> <li>[22]</li> <li>[23]</li> <li>[21]</li> <li>[2]</li> <li>[3]</li> </ul>	count.total h <integer> 653 643 638 758 788 913 711 393 792  1013 972 728 611 691 713 436 525 577 mean.quality <numeric> <na> 27</na></numeric></integer>	<pre>igh.quality high <integer> 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</integer></pre>	<pre>count. <integ< pre=""></integ<></pre>	<pre>y.ref hi eger&gt;     416     493     454     621     668     836     593     327     712      842     828     577     543     657     662     326     437     473 pos cour er&gt;     23     8     7 </pre>	nt.pos.ref <integer> 444 258 270</integer>	r.total hteger> 416 493 455 621 668 836 593 327 712  842 828 577 543 657 662 327 437 473 count.neg <integer> 0 0 34</integer>

[4]	<na></na>	34.83575	0	380	12
[5]	<na></na>	36.20060	8	317	0
[6]	<na></na>	35.49282	7	331	0
[7]	<na></na>	34.29174	0	379	9
[8]	<na></na>	35.21407	0	290	6
[9]	<na></na>	36.63062	0	526	7
 [16]	· · ·			 EE/	
[10]		34.00027	0	504	9
		34.34038	0	520 205	9
[10]		30.0/000	0	305	14
[19]		30.000//	4	110	0
[20]		30.33312	4 E	117	0
[21]	NA/ 27	22 5227/	1	212	17
[22]		25 10015	1 /	212	17
[23]		34 08457	11	207	0
[27]	count neg ref	cvcleCount 0 10	cvcleCount 10.90	cvcleCo	$\frac{1}{100}$
	<pre>countineg.ref <integer></integer></pre>	<pre><integer></integer></pre>	<pre><integer></integer></pre>	cycrcoo	<integer></integer>
[1]	185	(1100g01)	19		4
[2]	376	0	7		- 1
[3]	326	0	19		- 22
[4]	364	0	4		8
[5]	461	0	6		2
[6]	574	0	6		1
[7]	320	0	9		0
[8]	95	0	4		2
[9]	252	0	6		1
• • •					
[16]	448	0	4		5
[17]	433	0	9		0
[18]	407	0	5		9
[19]	525	0	3		1
[20]	575	0	1		3
[21]	588	0	5		0
[22]	205	1	17		0
[23]	222	0	14		0
[24] 	278	0	11		0
seqle	engths:				
2025	5767				

We can also manipulate the filters that call the variants that have already passed the basic QA checks.

```
> calling.filters <- VariantCallingFilters()
> summary(calling.filters, qa.variants)
```

<initial></initial>	${\tt readCount}$	likelihoodRatio	<final></final>
1281	75	23	20

### 2.4 Extending and customizing the workflow

Since the built-in filters are implemented using *FilterRules*, it is easy to mix and match different filters, including those implemented externally to the *VariantTools* package. This is the primary means of extending and customizing the variant calling workflow.

## 3 Exporting the calls as VCF

VCF is a common file format for communicating variants. To export our variants to a VCF file, we first need to coerce the *GRanges* to a *VCF* object. Then, we use writeVcf from the *VariantAnnotation* package to write the file (indexing is highly recommended for large files).

```
> vcf <- variantGR2Vcf(called.variants, sample.id = "H1993",
+ project = "VariantTools_Vignette")
```

```
> writeVcf(vcf, "H1993.vcf", index = TRUE)
```