Introduction to Variant Calling

Michael Lawrence

June 24, 2014

Outline

Outline

Variant calls

Definition

- A variant call is a conclusion that there is a nucleotide difference vs. some reference at a given position in an individual genome or transcriptome,
- Usually accompanied by an estimate of variant frequency and some measure of confidence.

Use cases

DNA-seq: variants

- Genetic associations with disease
- Mutations in cancer
- Characterizing heterogeneous cell populations

RNA-seq: allele-specific expression

- Allelic imbalance, often differential
- Association with isoform usage (splicing QTLs)
- RNA editing (allele absent from genome)

ChIP-seq: allele-specific binding

Variant calls are more general than genotypes

Genotypes make additional assumptions

- A genotype identifies the set of alleles present at each locus.
- The number of alleles (the ploidy) is decided and fixed.
- Most genotyping algorithms output genotypes directly, under a blind diploid assumption and special consideration of SNPs and haplotypes.

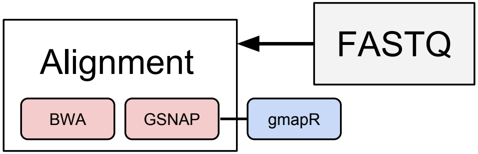
Those assumptions are not valid in general

- Non-genomic input (RNA-seq) does not represent a genotype.
- Cancer genome samples are subject to:
  - Copy number changes
  - Tumor heterogeneity
  - Tumor/normal contamination

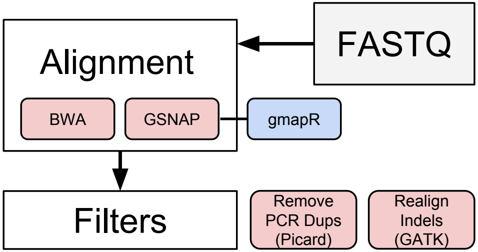
  So there is a mixture of potentially non-diploid genotypes, and there is no interpretable genotype for the sample

Typical variant calling workflow

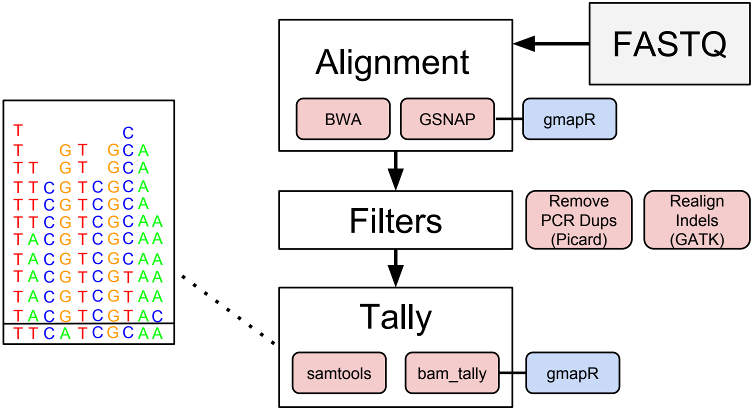
Typical variant calling workflow



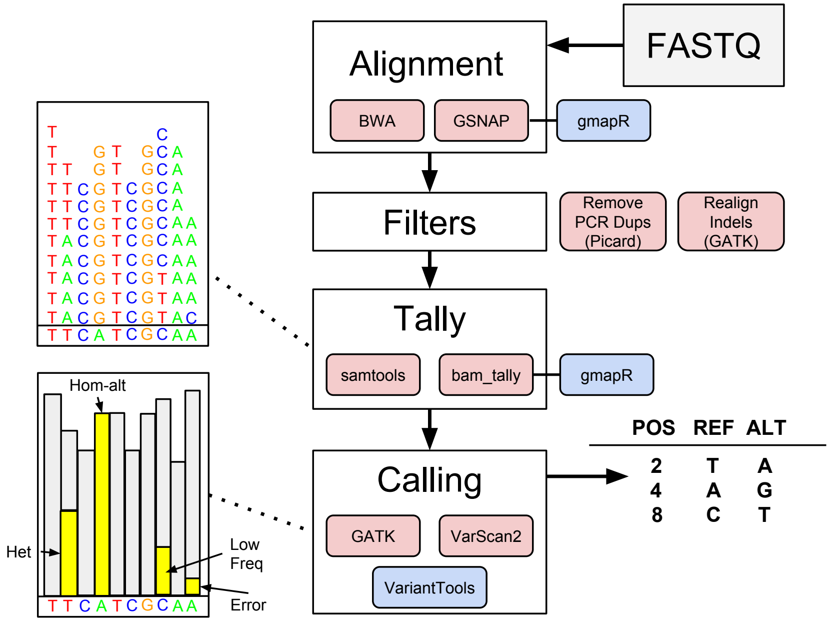
Typical variant calling workflow



Typical variant calling workflow

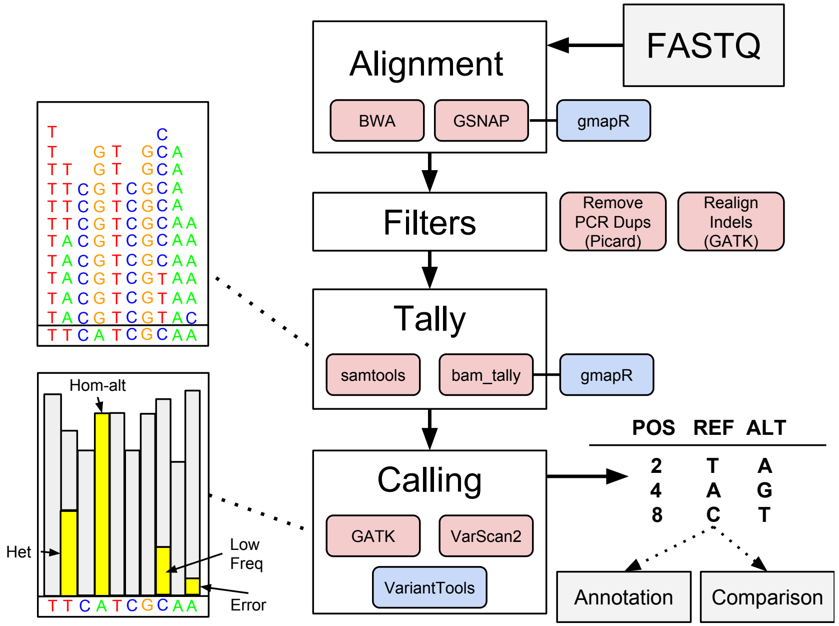


Typical variant calling workflow



POS	REF	ALT
2	T	A
4	A	G
8	C	T

Typical variant calling workflow



POS	REF	ALT
2	T	A
4	A	G
8	C	T

Sources of technical error

Errors can occur at each stage of data generation:

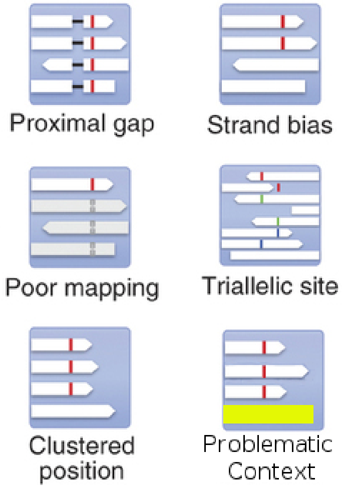
- Library prep
- Sequencing
- Alignment

Variant information for filtering

Information we know about each variant, and how it is useful:

Information	Utility
Base Qualities	Low quality indicates sequencing error
Read Positions	Bias indicates mapping issues
Genomic Strand	Bias indicates mapping issues
Genomic Position	PCR dupes; self-chain, homopolymers
Mapping Info	Aligner-dependent quality score/flags

Typical QC filters



10.1038/nbt.2514

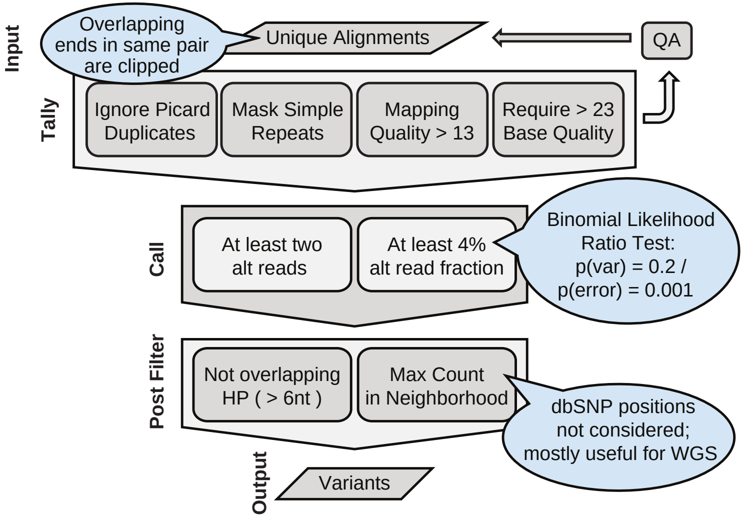
These filters are heuristics that aim to reduce the FDR; however, they will also generate false negatives and are best applied as soft filters (annotations).

Whole-genome sequencing and problematic regions

- Many genomic regions are inherently difficult to interpret.
  - Including homopolymers, simple repeats
- These will complicate the analysis with little compensating benefit and should usually be excluded.

Outline

VariantTools pipeline

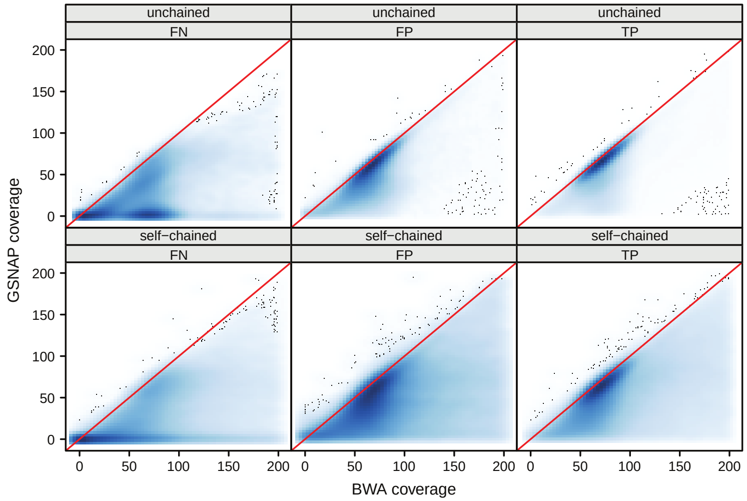


Input
Overlapping ends in same pair are clipped
Unique Alignments
QA
Tally
Ignore Picard Duplicates
Mask Simple Repeats
Mapping Quality > 13
Require > 23 Base Quality
Call
At least two alt reads
At least 4% alt read fraction
Binomial Likelihood Ratio Test: p(var) = 0.2 / p(error) = 0.001
Post Filter
Not overlapping HP (> 6nt)
Max Count in Neighborhood
dbSNP positions not considered; mostly useful for WGS
Output
Variants

UCSC self-chain as indicator of mappability

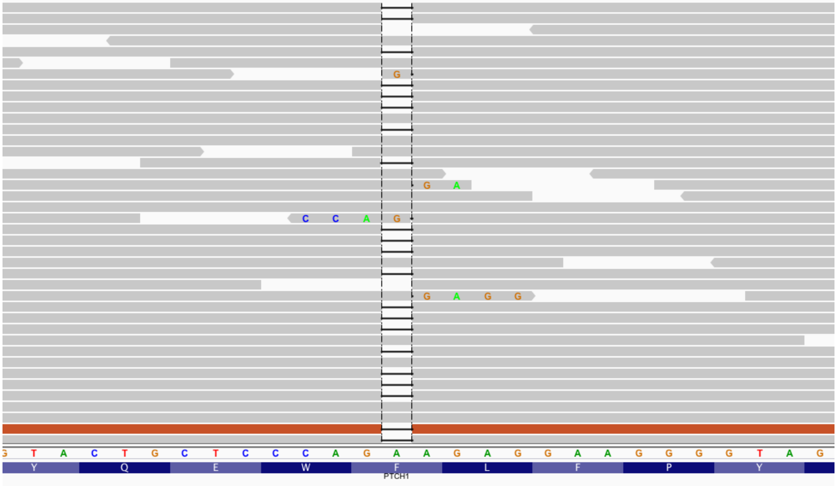
- UCSC publishes the self-chain score as a generic indicator of intragenomic similarity that is independent of any aligner
- About 6% of the genome fits this definition
- Virtually all (GSNAP) multi-mapping is in self-chains
- Lower unique coverage in self-chains

Aligner matters: coverage and mappability

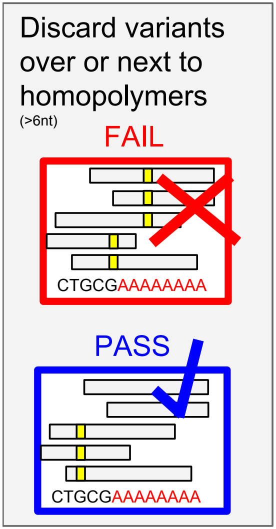


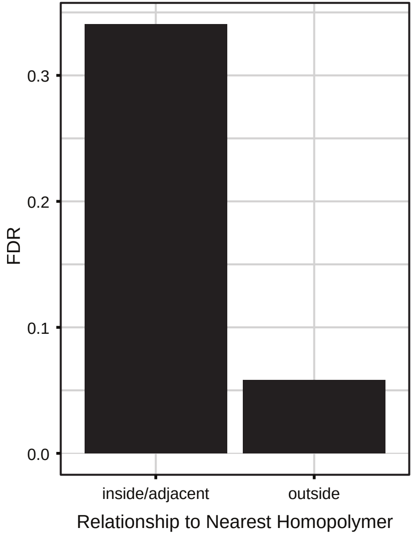
Aligning indels is error prone

Resolved by indel realignment



Homopolymers are problematic

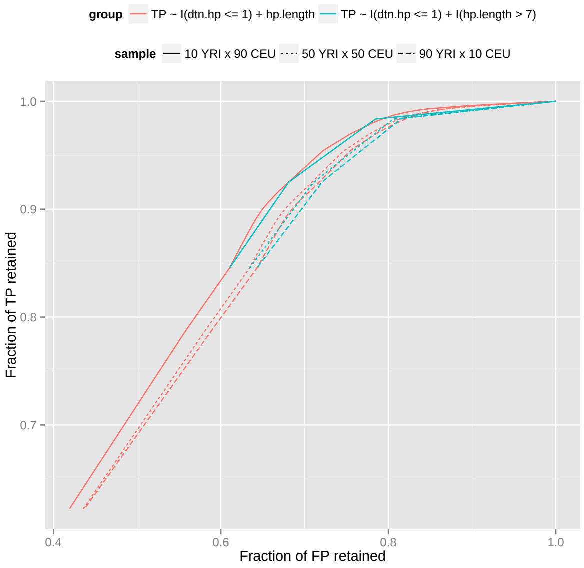




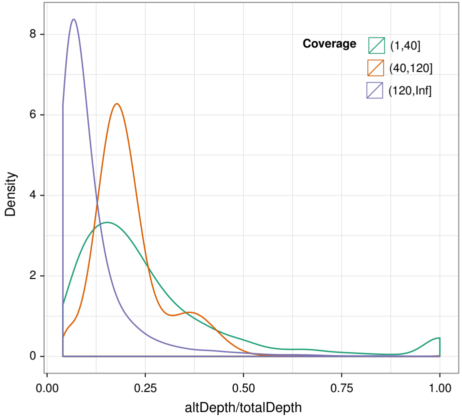
Choosing the homopolymer length cutoff

- We fit two logistic regressions to find the optimal length cutoff for our filter
- Response, *TP*: whether the variant call is a true positive
- Length as linear predictor:
  - *TP ~ I(hp.dtn <= 1) + hp.length*
- Indicator for when length exceeds 7:
  - *TP ~ I(hp.dtn <= 1) + I(hp.length > 7)*

Logistic regression results

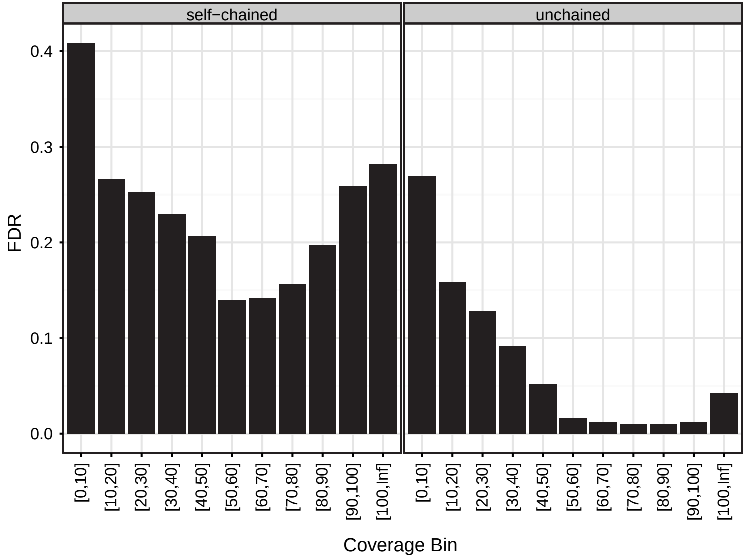


Effect of coverage extremes on frequencies

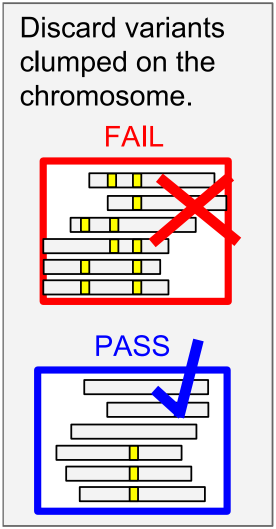


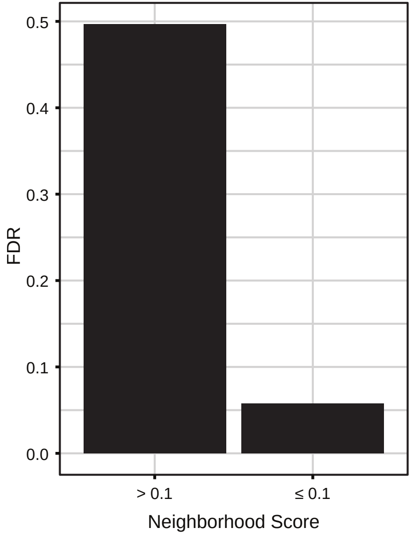
- Coverage sweet-spot (40-120) matches expected distribution.
- High coverage (>120) has much lower frequencies than expected; mapping error?
- Low coverage also different

Coverage extremes and self-chained regions



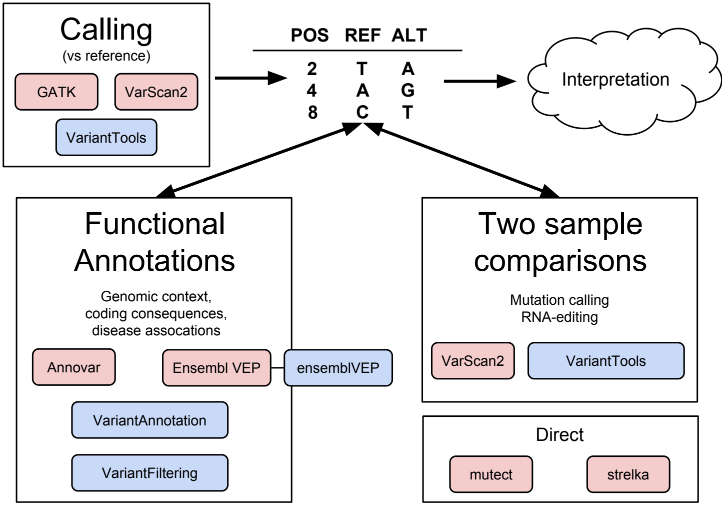
Variant density filter performance





Outline

Downstream of variant calling



Calling mutations through filtering

- We have two sets of variant calls (vs. reference) and need to decide which are specific to one (i.e., the tumor)
- We have to decide whether the variant frequency is:
  - Non-zero in tumor but
  - Zero in normal
- Variant frequencies are a function of:
  - Copy number changes
  - Tumor/normal contamination
  - Sub-clonality (tumor heterogeneity)
  - Mutations
- Mutations often present at low frequency and may even show up in the normal data due to contamination

VariantTools mutation calling algorithm

A mutation must pass the following filters:

- The variant was only called in the tumor
- There was sufficient coverage in normal to detect a variant, assuming the likelihood ratio model and given a power cutoff
- The raw frequency in normal is sufficiently lower than the frequency in tumor (avoids near-misses in normal)

Functional annotations with VariantAnnotation

The VariantAnnotation package

- Handles import/export of variants from/to VCF
- Defines central data structures for representing variants
  - *VCF* objects represent full complexity of VCF as a derivative of *SummarizedExperiment*
  - *VRanges* extends *GRanges* for special handling of variants
- Annotates variants with:
  - Genomic context: `locateVariants()`
  - Coding consequences: `predictCoding()`
  - SIFT/PolyPhen
- Filters VCF files as a stream (`filterVcf()`)

Learn more

Thursday lab on annotating variants

Outline

Overview

- Convenient interface for tallying mismatches and indels
- Several built-in variant filters
- Combines filters into a default calling algorithm
- Other utilities: call wildtype, ID verification
- Integrates:
  - *VRanges* data structure from `VariantAnnotation`
  - Tallying with `bam_tally` via gmapR
  - *FilterRules* framework from `IRanges`

Tallying

The underlying bam_tally from Tom Wu's GSTRUCT accepts a number of parameters, which we specify as a *TallyVariantsParam* object. The genome is required; we also mask out the repeats.

```
library(VariantTools)
data(repeats, package = "VariantToolsTutorial")
param <- TallyVariantsParam(TP53Genome(), mask = repeats)
```

Tallies are generated via the tallyVariants function:

```
tallies <- tallyVariants(bam, param)
```

VRanges

- The tally results are stored in a *VRanges* object
- Extension of *GRanges* to describe variants
- One element/row per position + alt combination
- Adds these fixed columns:

`ref`	ref allele
`alt`	alt allele
`totalDepth`	total read depth
`refDepth`	ref allele read depth
`altDepth`	alt allele read depth
`sampleNames`	sample identifiers
`softFilterMatrix`	*FilterMatrix* of filter results
`hardFilters`	*FilterRules* used to subset object

VRanges features

- Rough, lossy, two-way conversion between *VCF* and *VRanges*
- Matching/set operations by position and alt (`match`, `%in%`)
- Recurrence across samples (`tabulate`)
- Provenance tracking of applied hard filters
- Convenient summaries of soft filter results (*FilterMatrix*)
- Lift-over across genome builds (`liftOver`)
- *VRangesList*, stackable into a *VRanges* by sample
- All of the features of *GRanges* (overlap, etc)

Tally statistics

In addition to the alleles and read depths, `tallyVariants` provides:

Raw counts	Count before quality filter for alt/ref/total
Mean quality	Mean base quality for alt/ref
Strand counts	Plus/minus counts for alt/ref
Uniq read pos	Number of unique read positions for alt/ref
Mean read pos	Mean read position (cycle) for alt/ref
Var read pos	Variance in read position for alt/ref
MDFNE	Median distance from nearest end for alt/ref
Read pos bins	Counts in user-defined read pos bins for alt

Filtering framework

VariantTools implements its filters within the *FilterRules* framework from IRanges. The default variant calling filters are constructed by `VariantCallingFilters`:

```
calling.filters <- VariantCallingFilters()
```

Post-filters are filters that attempt to remove anomalies from the called variants:

```
post.filters <- VariantPostFilters()
```

Filter tallies into variant calls

The filters are then passed to the `callVariants` function:

```
variants <- callVariants(tallies, calling.filters,
                         post.filters)
```

Or more simply in this case:

```
variants <- callVariants(tallies)
```

Interoperability via VCF

We can export the variant calls to a VCF file:

```
writeVcf(variants, "variants.vcf", index = TRUE)
```

Outline

Visualizing variants with IGV SRAdb

Creating a connection to IGV

```
library(SRAdb)
startIGV("lm")
sock <- IGVsocket()
```

Exporting our calls as VCF

```
vcf <- writeVcf(variants, "variants.vcf", index = TRUE)
```

Creating an IGV session

Create an IGV session with our VCF, BAMs and custom p53 genome:

```
rtracklayer::export(genome, "genome.fa")
session <- IGVsession(c(bam.paths, vcf), "session.xml",
                      "genome.fa")
```

Load the session:

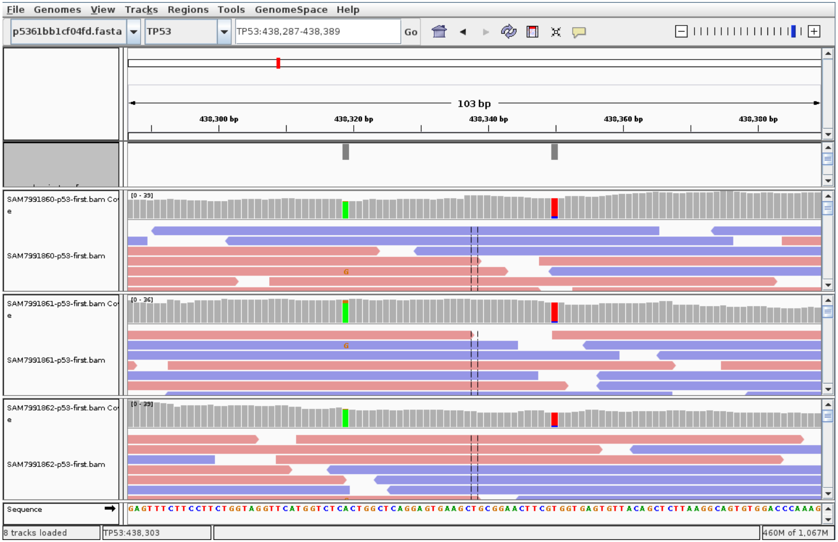
```
IGVload(sock, session)
```

Browsing regions of interest

IGV will (manually) load BED files as a list of bookmarks:

```r
rtracklayer::export(interesting.variants, "bookmarks.bed")
```

IGV section, from R


File Genomes View Tracks Regions Tools GenomeSpace Help
p5361bb1cf04fd.fasta
TP53
TP53:438,287-438,389
Go
103 bp
438,300 bp
438,320 bp
438,340 bp
438,360 bp
438,380 bp
SAM7991860-p53-first.bam Coverage
[0 - 39]
SAM7991860-p53-first.bam
G
SAM7991861-p53-first.bam Coverage
[0 - 36]
SAM7991861-p53-first.bam
G
SAM7991862-p53-first.bam Coverage
[0 - 39]
SAM7991862-p53-first.bam
Sequence
GAGTTTCTTCCTTCTGGTAGGTTCATGGTCTCACTGGCTCAGGAGTGAAGCTGCGGAACTTCGTGGTGAGTGTTACAGCTCTTAAGGCAGTGTGGACCCAAAG
8 tracks loaded
TP53:438,303
460M of 1,067M

VariantExplorer package

- The VariantExplorer package by Julian Gehring is an unreleased package for visually diagnosing variant calls
- Produces static ggbio plots and interactive web-based plots based on epivizr
- The epivizr package (Hector Corrada Bravo) is a browser-based genomic visualization platform that pulls data directly from a running R session
- Get epivizr:

```
devtools::install_github("epivizr", "epiviz")
```