`SVMerge (v1.2)` Pipeline Documentation

August 8, 2012

Contents

Introduction

SVMerge is a pipeline to call and computationally validate large structural variation (SV) calls generated from several software, which use different analysis methods and algorithms to identify candidate SVs. SVMerge can be used to set up a new SV calling project, run SV callers, merge redundant calls, set up and run local assemblies, align resultant contigs, parse and interpret contig alignments to validate SV calls and adjust breakpoints.

SVMerge is freely available for download at http://svmerge.sourceforge.net.

Reference: Wong K, Keane TM, Stalker J, Adams DJ, **Enhanced structural variant and breakpoint detection using SVMerge by integration of multiple detection methods and local assembly** Genome Biol. 2010;11(12):R128. Epub 2010 Dec 31. http://genomebiology.com/content/11/12/R128

Notes

The pipeline scripts are written in Perl and bash. All SVMerge scripts must be maintained in the same directory (eg: /home/user/src/svmerge/).

Along with SVMerge and the additional software listed below, the only requirements are BAM files, reference FASTA files, and a configuration file which specifies all parameters used in SVMerge. A template configuration file is included in this package.

Ideally, particular steps in the SVMerge pipeline should be run using a compute farm, especially with large data sets. SVMerge provides scripts to submit jobs jobs using the Platform LSF cluster management system (LSF) or Grid Engine. Commands to run specific steps without a farm are also provided.

Any file with coordinates for SVs or any other genomic features should be in tab-delimited or BED format (http://genome.ucsc.edu/goldenPath/help/hgTracksHelp.html#BED). This is required since BEDTools (see Additional Software section below) is used in the pipeline. Chromosomes can be either {1, 2, ... , X, Y} or {chr1, chr2, ... , chrX, chrY}, however the same naming convention *must* be used in all .tab or .bed files. These should also be consistent with chromosome names in your BAM and reference FASTA files.

A 'Quick Reference' guide and configuration file parameter list are located at the end of this document.

Additional software

Additional software required for `SVMerge` are freely available. Currently, `SVMerge` supports SV callers `BreakDancerMax, Pindel, cnD` and `SECluster`. These are available at:

https://trac.nbic.nl/pindel/downloads (Pindel_source_v0.2.3.zip)

http://sourceforge.net/projects/breakdancer/

http://sourceforge.net/projects/rdxplorer/ (No longer supported)

http://www.sanger.ac.uk/resources/software/cnd/

`SECluster` is included in the `SVMerge` package.

The BEDTools package is required for coordinate comparison:

http://code.google.com/p/bedtools/

The `Samtools` package is required to read and manipulate `BAM` files:

http://sourceforge.net/projects/samtools/

Note that if using `cnD`, the version of `samtools` you are using must have the `'pileup'` option.

The aligner for the contig alignments in the local assembly validation step is `Exonerate`:

http://www.ebi.ac.uk/~guy/exonerate/

The pipeline scripts also use the `Perl` module `Set::IntSpan`, which can be downloaded from `CPAN`:

http://search.cpan.org/dist/Set-IntSpan/

1 Configuration file

The configuration file is used throughout the pipeline, and specifies user parameters for the various stages of the pipeline. A template is provided with this package, `example.config`. Copy this file to your project directory, and edit as required. The parameters provided are suggested parameters for a human genome with approximately 40x mapped sequence coverage. Any parameter not applicable can be commented out with `#` at the beginning of the line.

To aid in set up of the configuration file, the following sections walk through the pipeline steps, and describe the relevant parameters. Ideally, all parameters should be set prior to starting a new project.

A complete list of the configuration file parameters is also available on the last page of this document.

2 Set up a new project

`SVMerge` requires a specific file structure in order for each step to find the appropriate files. The file structure is as follows:

```
/project/version/svcaller/logfiles/
/project/version/sv_call_Date/svcaller/
/project/version/sv_call_Date/merge/
/project/version/sv_call_Date/final/
/project/version/sv_call_Date/localAssemblies/config/
```

where `svcaller` is `breakdancer, pindel, etc.`

To set up a new project, the following parameters must be set in your configuration file:

```
project=NA18506                    # The name of your project directory
name=NA18506                       # Sample name to be used in output files
version=REL-01                     # Subdirectory for SV calling analysis
svdir=sv_calls_Jul2910             # Subdirectory for filtering and merging
chrRange=1-22                      # Chromosomes to be analyzed (numeric)
chrOther=X Y                       # Any other chromosomes [optional]
gender=male                        # Gender, if applicable
projdir=/full/path/to/project/NA18506      # Full project path
```

If your reference chromosomes are not numeric, comment the `chrRange` option and list all the chromosomes under `chrOther`, separated by `space`.

To indicate the SV callers you are using, specify the following:

```
callerlist=pindel sec breakdancer rdx (space-delimited)
```

```
The tags for the caller are:
breakdancer
pindel
sec
rdx
cnd
```

Choose a directory to create your project directory. This will be referred to as your '`main working directory`'. In this directory, run the following:

```
$EXEDIR/makeNewProject.sh configfile
```

where `$EXEDIR` is the location of the `SVMerge` package and `configfile` is your `SVMerge` configuration file. This will create a directory in your `main working directory` with the project name given in your config file, and all necessary subdirectories that will be required for downstream pipeline steps.

The location of your reference sequence files must also be indicated in your `SVMerge` configuration file.

```
bam=/path/to/file.bam                    # Full path to BAM file
bai=/path/to/file.bam.bai                # Full path to BAM index file
bamdir=/path/to/bamDirectory/            # Directory containing chromosome BAMs
chrrefdir=/path/to/ref/chromDir/         # Dirctory containing chromosome FASTAs
reffile=/path/to/ref/ref.fa              # FASTA file with reference chromosomes
```

Your data may be in a single `BAM` file containing all chromosomes (specify `bam` and `bai`, comment out `bamdir` with `#bamdir`), or may be split up by chromosome into different `BAMs` (specify `bamdir`, comment out `bam` and `bai`. If you have one `BAM` per chromosome, they must be named by the chromosome name only, eg: `1.bam, 2.bam, 3.bam, ... , X.bam, Y.bam`.

Note that all `BAM` files *must* be indexed; the index files are named with a `.bai` extension, eg: `sample.bam.bai` is the index file for `sample.bam`. Refer to the `Samtools` package for further details.

Your reference chromosomes may also be in a single `FASTA` file (`reffile`) or in separate `FASTA` files, in a single directory `chrrefdir`).

3 Run the SV callers

The raw output for the SV callers will be in directories:

```
/project/version/breakdancer/
/project/version/pindel/
/project/version/rdx/
/project/version/etc/
```

The directories created will depend on the specific callers listed in the configuration file. Other SV calls may be included for analysis in subsequent steps by placing the tab-delimited SV calls, separated by SV type, in new directories. For example:

```
/project/version/newcaller/del.tab
/project/version/newcaller/ins.tab
/project/version/newcaller/inv.tab
/project/version/newcaller/gain.tab
/project/version/newcaller/loss.tab

column format:
chr   SVstart   SVend   Annotation

Annotation has the format:
SVType_SVcaller_SAMPLENAME_Size

eg:
DEL_RP_HUMAN_3402
INS_RP_HUMAN_303
INV_RP_HUMAN_5030
GAIN_RP_HUMAN_10400
LOSS_RP_HUMAN_30000
```

The files must be named `del.txt, ins.txt, etc.` See `Section 3.2.6` for futher details.

3.1 Using a compute farm

If you have access to a compute farm, a `Perl` script is provided to submit SV caller jobs in jobs arrays, creating the output in the appropriate directories.

From your `main working directory`, run:

```
$EXEDIR/runSVcallers.pl -c configfile -r runVersion -j [LSF|SGE] [-v]
```

You will need to give the location of the software you are using:

```
exedir=/path/to/SVMerge/                # SVMerge package location
samtools=/path/to/samtools              # samtools binary
cnddir=/path/to/cnD/                    # cnD directory
bam2conf=/path/to/bam2cfg.pl            # BreakDancer config file generator
bdexe=/path/to/BreakDancerMax.pl        # BreakDancer Perl script
pinexe=/path/to/pindel_x86_64           # Pindel binary
secexe=/path/to/SECluster.pl            # Provided in the SVMerge package
exonerateExe=/path/to/exonerate         # Exonerate binary
```

The following parameters are set in the `SVMerge` configuration file:

```
defaultQueue=normal                     # Default queue name, if none specified below

BreakDancerMax:
BDconf=1                                # Run bam2cfg.pl to create config
BDconfParams=-c 7 -n 10000              # Parameters for bam2cfg.pl
BDparams=-q 20 -c 7                     # Parameters for BreakdancerMax
BDcopynum=2                             # Copynumber cutoff for deletions, using the
                                          Breakdancer estimate (column 12)
BDmem=3000                              # Memory usage [optional]
BDqueue=normal                          # Farm queue for BreakDancerMax [optional]

Pindel:
PDconf=/path/to/pindel.conf             # Pindel config file (bams and insert sizes)
PDfiltermem=3000                        # Memory usage for PDgetReads
PDmem=2000                              # Pindel memory usage
PDqueue=normal                          # Farm queue for Pindel [optional]

SECluster:
SECfilter=1                             # Extract reads from BAMs for SECluster
SECqual=20                              # Quality cutoff of reads
SECmin=5                                # Minimum reads in either the forward
                                          or reverse cluster, when f and r
                                          clusters are paired
SECminCluster=3                         # Minimum reads to form a single end forward
                                          or reverse cluster
SECmax=500                              # Maximum reads to form a cluster
SECfilterQueue=normal                   # Farm queue for SECfilter [optional]
SECfilterMem=2000                       # Memory usage for SECfilter [optional]
SECqueue=normal                         # Farm queue for SEC [optional]
SECmem=4000                             # Memory usage for SEC [optional]

cnD:
CNDpileup=1                             # Run initial samtools pileup step
CNDsnprate=0.001                        # Expected SNP rate
CNDparams=--repeat-cutoff=0.35          # Parameters for cnD
CNDnohet=1                              # Do not call heterozygous CN losses
CNDgccorrect=1                          # Run GC correction on read depth
CNDpileupMem=3000                       # Farm queue for CNDpileup [optional]
CNDpileupQueue=normal                   # Memory usage for CNDpileup [optional]
CNDmem=2000                             # Farm queue for cnD [optional]
CNDqueue=normal                         # Memory usage for cnD [optional]
```

```
RDXplorer:
RDXsplitBam=1                          # Create chrom. BAMs from a single BAM
RDXqueue=long                          # Farm queue for RDXplorer [optional]
RDXmem=2000                            # Memory usage for RDXplorer [optional]
```

3.2 Running SV callers without using `runSVcallers.pl`

If not using `runSVcallers.pl`, the output must follow a specific naming convention in order to be recognized by the downstream pipeline scripts. Details for the SV callers are below, however, you will need to refer to the documentation for each SV caller for more details.

3.2.1 BreakDancerMax

To run `BreakDancerMax`, a configuration file is first produced for each `BAM`. For example:

```
Single BAM

bam2cfg.pl -q [BDmapq] -n [readsToSample] bamfile > bd.config

Chromosome BAMs

bam2cfg.pl -q [BDmapq] -n [readsToSample] chr.bamfile > bd.chr.config
```

where `BDmapq` is the minimum read mapping quality, and `readsToSample` is the number of reads used to calculate the insert size distribution. See BreakDancer documentation for more options. If each chromosome is in a separate `BAM` file, a configuration file must be created for each chromosome `BAM`.

For each chromosome, run BreakDancerMax using:

```
BreakDancerMax.pl -f -o [chr#] -q [BDmapq] bd.config > name.chr#.max

or:

BreakDancerMax.pl -f -o [chr#] -q [BDmapq] bd.chr.config > name.chr#.max
```

There must be one output file per chromosome, and the files named:

```
name.chr#.max
```

where `name` is the sample name specified in your `SVMerge` configuration file, and `chr` is each chromosome in `chrRange` and `chrOther`.

3.2.2 Pindel

SVmerge supports `Pindel v0.2.3` and higher, which takes in a `bam` file as input and a configuration file which lists your bams, library insert sizes and sample names. See `Pindel` documentation for more details.

For each chromosome and `BAM` file, run:

```
pindel -f ref.fa -i pindel_conf -c chrom -o prefix [optional_parameters]
```

The `ref.fa` file is a `FASTA` file containing the reference sequence, and `chrom` is the chromosome name and `pindel_conf` is the configuration file. To use the output file in downstream steps, the prefix must be the sample `name`, eg: NA18507.

3.2.3 SECluster

`SECluster` uses paired-end reads with one end mapped and one end unmapped to find candidate large insertions. To filter out these reads from your `BAM` file(s), for each chromosome, run:

```
$EXEDIR/samfilter.sh bamfile [chromsome] 1
```

or:

```
$EXEDIR/samfilter.sh chr.bamfile [chromosome] 1
```

This produces files named `chr.se.sam` for each chromosome. These are used as input for SECluster:

```
SECluster.pl -f chr.se.sam -q [quality] -m [minReads] -c [minReads] -r [chr] \
    -x [maxReads] > name.chr.clusters
```

where

```
q=read mapping quality [20 is recommended]
m=minimum reads to form a cluster [5 is recommended]
c=minimum reads in each forward and reverse cluster [5 is recommended]
x=maximum reads in a cluster [depends on depth of data set]
```

and `name` is the sample name in your configuration file. The output files must be named in the format `name.chr.clusters`.

3.2.4 RDXplorer

NOTE: the version of RDXplorer compatible with SVMerge is no longer available, and we have not yet updated SVMerge to run with the version available on the RDXplorer website.

RDXplorer requires all BAM files to be placed in the same directory. If each chromosome is already in a separate BAM file, you can create symbolic links to these files in the /project/version/rdx/ directory. The links should be named with the format:

```
name.chrom[chr].bam
```

```
eg: NA18506.chrom1.bam
```

If your data set is high depth and in a single BAM, you can use samtools to create a BAM for each chromosome:

```
samtools view -b bamfile [chr] > name.chrom[chr].bam
```

Adjust the necessary parameters in the RDXplorer script run.sh, and in your /project/version/rdx/ directory run:

```
/path/to/rdx/run.sh
```

Note that RDXplorer currently only works for human data sets.

3.2.5 cnD

cnD requires several steps to run; refer to the documentation. cnD is designed to call copy number gain and loss in homozygous genomes, such as inbred mice. The final output should be in the /project/version/cnd/ directory and the files should be named chr.calls for each chromosome. Note that cnD required samtools pileup which is deprecated in newer versions of samtools.

3.2.6 User-defined calls

User-defined calls may be incorporated in the pipeline if formatted in tab-delimited or BED format. The files must be indicated in the SVMerge configuration file:

```
otherDelCalls=caller1/del.tab caller2/del.tab # Del. with read pair (RP) support
otherLossCalls=caller3/loss.tab               # CN loss (del. with no RP support)
otherInsCalls=caller1/ins.tab
otherInvCalls=caller3/inv.tab
otherGainCalls=caller2/gain.tab caller3/gain.tab
```

where caller1, caller2, etc. are directory names for additional SV callers (separated by space), and SV calls are separated by SV type (del, ins, inv, gain). All calls can be filtered and merged as usual (see 'Filter and merge calls' section). Columns 1 to 3 are the coordinates of the SV call:

```
chr   SVstart   SVend
```

where `chr` is the reference chromosome.

The fourth column in the `tab` files must have the following format:

```
SVType_SVcaller_SAMPLENAME_Size

eg:
DEL_RP_HUMAN_3402
INS_RP_HUMAN_303
INV_RP_HUMAN_5030
GAIN_RP_HUMAN_10400
LOSS_RP_HUMAN_30000
```

4 Filter and merge calls

Once all SV callers have run to completion, optional, additional filtering may be applied. There are two filtering options, one for filtering calls by score (if applicable), and another for filtering by genomic location. Calls near or overlapping reference sequence assembly gaps, centromeres, and telomeres are likely to be artifacts, and the user may wish to remove these. You will need to download these files for your reference genome, and format them, if necessary, into tab-delimited format to be recognized by `BEDTools`:

```
chr   startCoord   endCoord   annotation
```

where `chr` is the chromosome name.

If no filtering options are set, the calls will simply be merged into a non-redundant set of SV calls.

The following parameters may be set in the configuration file:

Filtering by score:

```
BDscore=25                              # BreakDancerMax score cutoff
BDrs=2                                  # BreakDancerMax min. supporting RPs
PDscore=30                              # Pindel score cutoff
PDsupports=10                           # Pindel min. supporting reads
RDXscore=5                              # RDXplorer Z-score cutoff
```

Filtering by location:

```
bedexe=/path/to/BEDTools/bin/intersectBed       # BEDTools intersect binary
overlapexe=/path/to/SVMerge/overlapExons.pl     # Included with SVMerge
gaps=/path/to/hg19_gap.tab                      # Sequence gaps to avoid
gapsBuffer=600                                  # Distance from sequence gaps
centel=/path/to/hg19_cen_tel.tab                # Centromere and telomeres
centelBuffer=1000000                            # Distance from centromeres
                                                  and telomeres

gapOverlap=0.25                                 # Fraction that must overlap
```

```
                                                  with gap/cen/tel
filterOther=/path/to/regionsToAvoid.tab          # Other regions to avoid
filterOtherBuffer=200                            # Buffer for 'filterOther'
```

All user-defined SV calls (`otherDelCalls, otherLossCalls, otherInsCalls, otherInvCalls, otherGainCalls`) will also be filtered for overlapping calls within the same `*.tab` file, and calls <100bp are removed. The option to apply filtering by location to user-defined calls, set by the above parameters, is indicated by the parameter:

```
otherFilterGaps=1                # Apply location filtering to user-defined calls
```

From your `main working directory`, run:

```
$EXE/filterAndMerge.pl -c configfile
```

This creates the `'merged raw'` SV call set:

```
/project/version/svdir/merged/name.merged.tab
```

This is the `'merged raw'` call set; it is a tab-delimited file. Columns 1 to 3 give the coordinates, and column 4 provides information about the SV call.

```
1 1925118 1925216 INS_merge_NA18506
1 3129140 3129414 INS_merge_NA18506_136
1 1162695 1162838 DEL_merge_NA18506_163
1 104455176 104457924 INV_BD_NA18506_2667
1 83824901 83835300 LOSS_RDX1_NA18506_10400
...

INS/DEL/INV/LOSS/GAIN = SV type (Losses are deletions without RP support)
merge/BD/RDX1/etc. = SV caller; merge results from merging 2 or more call sets
                     BD is BreakDancer, RDX1 is RDXplorer, copy number 1
NA18506 = project name
136/163/etc. = SV size. No size estimate is provided for some large insertions
```

A `BED` formatted file is also created for the SV call list, which may be uploaded to the UCSC Genome Browser (http://genome.ucsc.edu/cgi-bin/hgGateway), if your reference genome is available. SV call types are color-coded. The file created is:

```
/project/version/svdir/merged/name.ALL.merged.bed
```

5 *De novo* local assemblies and alignments

Where applicable, SV calls from the `'merged raw'` call list are computationally validated by local assembly. This applies to calls which have evidence from read-pair analysis or split-mapping. Calls derived from read-depth alone are not subjected to local assembly, but are retained without further evaluation for the `'final' SVMerge` call set. Currently, Velvet and ABySS are supported in `SVMerge`.

5.1 Running local assemblies and alignments using a compute farm

You can submit jobs with the `runAssembly.pl` script provided in `SVMerge`. The following parameters can be set:

```
makeConfig=1                                # Create assembly config files from
                                              name.merged.tab
chrrefdir=/path/to/chromosome/dir/          # Path to directory with chromosome
                                              FASTA files (or 'reffile')
reffile=/path/to/ref/ref.fa                 # FASTA file with reference chromosomes
submatrix=/path/to/SVMerge/submat.txt       # Provided with SVMerge package; used
                                              by Exonerate for checking inversions
joblimit=75                                 # Max. assembly jobs to run from a
                                              job array [optional]
checkdone=1                                 # Check for existing exonerate
                                              output and run only if it
                                              doesn't exist
subseq=1                                    # Align to a slice of the reference
                                              genome rather than the whole
                                              chromosome [recommended]
outdir=.                                    # Where to put the 'localAssemblies'
                                              directory if NOT in directory
                                              /project/version/svdir/ (it is not
                                              recommended to change this)
assemMin=1                                  # run jobs 1-100 only [optional]
assemMax=100
assemQueue=normal                           # Farm queue [optional]
assemMem=2000                               # Memory job requirement [optional]
```

The following `Velvet` parameters can be set:

```
velvet=1                                    # set to 1 if velvet is used
velveth=/path/to/velveth
velvetg=/path/to/velvetg
hashlen=29                                  # Hash length (k-mer size)
ins_len=220                                 # Library insert size
exp_cov=35                                  # Expected coverage
```

```
cov_cutoff=2                                    # Minimum coverage
```

The following ABySS parameters can be set:

```
abyss=1                                         # Use ABySS for assembly
abyss-pe=/path/to/abyss-pe                      # Path to ABySS-pe binary
kmer=25                                         # Hash length (k-mer size)
npairs=10                                       # Minimum read pairs for scaffolding
```

It should not be necessary to use both Velvet and ABySS for assembly. If using Velvet, set abyss=0 and *vice versa*.

From your main working directory, the following script can be used to submit job arrays:

```
$EXEDIR/runAssembly.pl -c configfile -s name.merged.tab -v runVersion -j [LSF|SGE]
```

where name.merged.tab is the 'merged raw' SV call set and runVersion is a unique number or ID for the job submission. This Perl script will submit farm jobs to create the necessary config files for local assembly, then runs the local assemblies and contig alignments.

5.2 Running local assemblies without using runAssembly.pl

The same configuration file parameters listed in the previous section apply. If you are not using a farm, the following steps must be run:

(a) Create configuration files:

Working directory:

```
/project/version/svdir/localAssemblies/config/
```

Run:

```
$EXEDIR/coord2config.pl -c configfile -b name.merged.tab
```

The output of this script are files named sv.config.1, sv.config.2, etc.

(b) Run local assemblies and alignments:

Working directory:

```
/project/version/svdir/localAssemblies/
```

Run:

```
$EXEDIR/svAssemble.pl config/sv.config.1 pathToSamtools [1]
$EXEDIR/svAssemble.pl config/sv.config.2 pathToSamtools [1]
```

...

where '1' is an option to run assemblies only for SV calls without existing `Exonerate` output.

6 Parse alignments

Once all assemblies and alignments are completed successfully, the alignments are parsed for contigs which provided evidence for SV breakpoints (eg: a contig with a deletion will align with a large gap; a contig with an insertion will either contain the whole insertion or part of the insertion). An additional read-depth check for deletions,using the `BAM` file(s), can also be performed (`bamcheck=1`). This is especially usefull when checking heterozygous deletions, where the reads from the non-deleted copy make local assembly confirmation more difficult.

The following parameters can be set:

```
bamcheck=1                  # Apply read-depth check for deletions
meanCov=42                  # Expected mapped coverage (use with
                              bamcheck=1)
zyg=het                     # Zygosity of your genome (het or hom), use
                              with bamcheck=1 for checking read-depth of
                              deletions
offset=1                    # Apply offset (1 if subseq=1 was used)
                              are many calls close together.
parseSplit=1                # Run splitFile.sh to split name.merged.tab
                              for the alignment parsing step
parseSplitLines=250         # Number of lines for each split name.merged.tab.*
parseJoblimit=10            # Maximum jobs in your array to run at simulaneously
parseQueue=normal           # LSF/SGE queue to submit alignment parse jobs
parseMem=2000               # Memory usage for alignment parse jobs
```

If using a compute farm, the from your `main working directory`, run:

```
$EXEDIR/runAlignParse.pl -c /full/path/to/configfile -s name.merged.tab  \
-d localAssembliesDir -v runVersion -j [LSF|SGE]
```

where `localAssembliesDir` is the location of the chromosome directories containing the results of the local assemblies (normally `/project/version/svdir/localAssembiles/`), and runVersion is a number or ID that makes the job submission ID unique. This produces output files `alignparse.1, alignparse.2, etc.` in the `localAssemblies` directory.

Otherwise, follow these steps:

(a) Prepare the input files

Working directory:

```
/project/version/svdir/localAssemblies/

$EXEDIR/splitFile.sh ../merged/name.merged.bed 500
```

where 500 is the number of lines (SV calls) per file. This splits up the SV calls in order to run jobs in parallel. The output files are called `name.merged.tab.1, name.merged.tab.2, etc`.

(b) Parse the alignments

Working directory:

```
/project/version/svdir/localAssemblies/

$EXEDIR/runParser.pl -s name.merged.tab.1 -c configfile > alignparse.1
$EXEDIR/runParser.pl -s name.merged.tab.2 -c configfile > alignparse.2
...
```

7 Interpret results

All parsed alignments must now be concatenated and interpreted. If using a compute farm, run the following script:

In your `main working directory`:

(a) Concatenate the files

Working directory:

```
/project/version/svdir/localAssemblies/

cat alignpares.* > alignparse
```

(b) Parse the output

Working directory:

```
/project/version/svdir

$EXEDIR/parseBoundary.pl -a localAssemblies/alignparse -o final/sv.final
```

The resulting files are created:

```
/project/version/svdir/final/

sv.final.rank1.tab          # Highest confidence set
```

```
sv.final.rank2.tab          # Local assembly results ambiguous
sv.final.rank3.tab          # Local assembly showed no breakpoints
sv.final.small.tab          # Local assembly SV found is <100bp
sv.final.query.tab          # SV is in a region with unusually high
                              coverage
```

8 Merge final call set

The SV calls in `/project/version/svdir/final/sv.final.rank1.tab` will make up the 'final' `SVMerge` call set. One final merging step, to find overlapping SV calls (complex SVs), must be run:

Working directory:

```
/project/version/svdir/final/

$EXEDIR/mergeFinalCalls.pl -f sv.final.rank1.tab -c configfile  \
> sv.final.rank1.merged.tab
```

This is the 'final' `SVMerge` call set. The output is in tab-delimited format, eg:

```
10 107253013 107253366 DEL_REF_10_107253004_353
10 109728588 109729513 DEL_RAW_10_109728588_925
10 101587707 101587708 INS_REF_10_101587682_1026
...
```

Columns 1 to 3 are the SV coordinates. Column 4 provides the following information:

```
DEL/INS      = the SV type*
REF/RAW      = coordinates refined from local assembly or the original coordinates
10_107253004 = chromosome and original SV 5' breakpoint coordinate (from the 'merged
               raw' file sample.merged.tab)
353          = SV size**
```

*SV types include `DEL/LOSS, INS/INSi, INV, GAIN` and more complex SVs which include two or more SV types: `DELINS, INVDEL, INVINS, INVCOMPLEX`.

**For insertions, if the complete insertion has been reconstructed, the SV type will be `INSi` and the size will be the size of the insertion. If only part of the insertion was reconstruced the size will reflect only the total size of the reconstructed regions.

To create a `BED` file from the final call set:

```
tab2bed.pl -f NA18506.merged.tab -n NA18506
```

will produce file 'NA18506.final.bed' with BED annotions:

```
track name="NA12830 final SVMerge"
description="NA12830 final SVMerge" useScore=0 itemRgb="On" visibility=2
chr19 1029070 1029554 DEL_SVMerge_NA12830_483 0 + 1029070 1029554 255,0,0 1 484 0
chr19 1034999 1035400 GAIN_SVMerge_NA12830_77251 0 + 1034999 1035400 34,139,34 1 401 0
...
```

Quick Reference Guide - Pipeline steps

Once your configuration file is set up, the `SVMerge` pipeline can be run with the following commands:

(a) Create a new project in your `main working directory`:

```
$EXEDIR/makeNewProject.sh configfile
```

where `$EXEDIR` is the location of the `SVMerge` package.

(b) Run SV callers

In your `main working directory`:

```
$EXEDIR/runSVcallers.pl -c configfile -r runVersion -j [LSF|SGE] [-v]
```

(c) Filter and merge:

In your `main working directory`:

```
$EXE/filterAndMerge.pl -c configfile
```

This creates the `'merged raw'` SV call set in /project/version/svdir/merged/name.merged.tab

(d) Run local assemblies and contig alignments

In your `main working directory`:

```
$EXEDIR/runAssembly.pl -c configfile -s name.merged.tab -v runVersion -j [LSF|SGE]
```

(e) Parse alignments

In your `main working directory`:

```
$EXEDIR/runAlignParse.pl -c /full/path/to/configfile -s name.merged.tab  \
-d localAssembliesDir -v runVersion -j [LSF|SGE]
```

(f) Interpret results

Working directory:

```
/project/version/svdir/}:

cat localAssemblies/alignparse.* > localAssemblies/alignparse
$EXEDIR/parseBoundary.pl -a localAssemblies/alignparse -o final/sv.final
```

From this directory:

```
/project/version/svdir/final/
```

Run:

```
$EXEDIR/mergeFinalCalls.pl -f sv.final.rank1.tab -c configfile  \
> sv.final.rank1.merged.tab
```

The file `sv.final.rank1.merged.tab` is the 'final' `SVMerge` call set.

Create a `BED` file from the final call set:

```
$EXEDIR/tab2bed.pl -f NA18506.merged.tab -n NA18506
```

Quick Reference Guide - pipeline steps (not using a compute farm)

Once your configuration file is set up, the `SVMerge` pipeline can be run with the following commands:

(a) Create a new project in your `main working directory`

```
$EXEDIR/makeNewProject.sh configfile
```

where `$EXEDIR` is the location of the `SVMerge` package.

(b) Run SV callers

See `Section 3.2`

(c) Filter and merge:

In your `main working directory`:

```
$EXE/filterAndMerge.pl -c configfile
```

This creates the `'merged raw'` SV call set in /project/version/svdir/merged/name.merged.tab

(d) Run local assemblies and contig alignments

First create the assembly configuration files:

Working directory:

```
/project/version/svdir/localAssemblies/config/
```

Run:

```
$EXEDIR/coord2config.pl -c configfile -b name.merged.tab
```

The output of this script are files named `sv.config.1, sv.config.2, etc.`

Submit assembly jobs for each sv.config.* file.

Working directory:

```
/project/version/svdir/localAssemblies/
```

Run:

```
$EXEDIR/svAssemble.pl config/sv.config.1 pathToSamtools [1]
$EXEDIR/svAssemble.pl config/sv.config.2 pathToSamtools [1]
...
```

(e) Parse alignments

First split the `name.merged.tab` file.

Working directory:

```
/project/version/svdir/localAssemblies/
```

Run:

```
$EXEDIR/splitFile.sh ../merged/name.merged.tab 500
```

where 500 is the number of lines per file. This will produce files called name.merged.tab.1, name.merged.tab.2, etc.

Use these files as input for the alignment parsing scripts.

Working directory:

```
/project/version/svdir/localAssemblies/
```

Run:

```
$EXEDIR/runParser.pl -s name.merged.tab.1 -c config/sv.config.1 > alignparse.1
$EXEDIR/runParser.pl -s name.merged.tab.2 -c config/sv.config.2 > alignparse.2
...
```

(f) Interpret results

Concatenate the `alignparse.*` files:

Working directory:

```
/project/version/svdir/localAssemblies/
```

```
cat alignparse.* > alignparse
```

Run script to parse and interpret the alignment results:

Working directory:

```
/project/version/svdir/
```

Run:

```
$EXEDIR/parseBoundary.pl -a localAssemblies/alignparse -o final/sv.final
```

Working directory:

```
/project/version/svdir/final/
```

Run:

```
$EXEDIR/mergeFinalCalls.pl -f sv.final.rank1.tab -c configfile  \
> sv.final.rank1.merged.tab
```

The file `sv.final.rank1.merged.tab` is the 'final' `SVMerge` call set.

Create a `BED` file from the final call set:

```
tab2bed.pl -f NA18506.merged.tab -n NA18506
```

Configuration file parameters

General parameters:

```
project=NA18506                          # The name of your project directory
name=NA18506                             # Sample name
version=REL-01                           # Subdirectory for SV calling analysis
svdir=sv_calls_Jul2910                   # Subdirectory for filtering and merging
chrRange=1-22                            # Chromosomes to be analyzed (numeric)
chrOther=X Y                             # Any other chromosomes
gender=male                              # Gender, if applicable
projdir=/full/path/to/project/NA18506    # Full working directory
defaultQueue=normal                      # Default queue for running LSF/SGE jobs
```

Location of software:

```
exedir=/path/to/SVMerge/                 # SVMerge package location
samtools=/path/to/samtools               # samtools binary
exonerateExe=/path/to/exonerate          # Exonerate binary
```

Location of reference files:

```
bam=/path/to/file.bam                    # Full path to BAM file
bai=/path/to/file.bam.bai                # Full path to BAM index file
bamdir=/path/to/bamDirectory/            # Directory containing chromosome BAMs
chrrefdir=/path/to/ref/chromDir/         # Directory containing chromosome FASTAs
reffile=/path/to/ref/ref.fa              # FASTA file with reference chromosomes
```

Location of SV callers (if using `runSVcallers.pl`):

```
cnddir=/path/to/cnDdir/                  # cnD directory
bam2conf=/path/to/bam2cfg.pl             # BreakDancer config file generator
bdexe=/path/to/BreakDancerMax.pl         # BreakDancer Perl script
pinexe=/path/to/pindel_x86_64            # Pindel executable
rdxdir=/path/to/rdxdir/                  # RDX directory
secexe=/path/to/SECluster.pl             # SECluster Perl script
```

SV callers used:

```
callerlist=pindel sec breakdancer cnd    # List of all SV callers used
```

SV caller parameters (if using `runSVcaller.pl`):

```
defaultQueue=normal                      # Default LSF/SGE queue name

BreakDancerMax:
BDconf=1                                 # Run bam2cfg.pl to create config
BDconfParams=-c 7 -n 10000               # Parameters for bam2cfg.pl
BDparams=-q 20 -c 7                      # Parameters for BreakdancerMax
```

```
BDcopynum=2                          # Copynumber cutoff for deletions, using
                                       the Breakdancer estimate (column 12)
BDmem=3000                           # Memory usage [optional]
BDqueue=normal                       # Farm queue for BreakDancerMax [optional]

Pindel:
PDconf=/path/to/pindel.conf          # Pindel config file (bams, insert sizes)
PDfiltermem=3000                     # Memory usage for PDgetReads
PDmem=2000                           # Pindel memory usage
PDqueue=normal                       # Farm queue for Pindel [optional]

SECluster:
SECfilter=1                          # Extract reads from BAMs for SECluster
SECqual=20                           # Quality cutoff of reads
SECmin=5                             # Minimum reads in either the forward or
                                       reverse cluster, when f and r clusters
                                       are paired
SECminCluster=3                      # Minimum reads to form a single end
                                       forward or reverse cluster
SECmax=500                           # Maximum reads to form a cluster
SECfilterQueue=normal                # Farm queue for SECfilter [optional]
SECfilterMem=2000                    # Memory usage for SECfilter [optional]
SECqueue=normal                      # Farm queue for SEC [optional]
SECmem=4000                          # Memory usage for SEC [optional]

cnD:
CNDpileup=1                          # Run initial samtools pileup step
CNDsnprate=0.001                     # Expected SNP rate
CNDparams=--repeat-cutoff=0.35       # Parameters for cnD
CNDnohet=1                           # Do not call heterozygous CN losses
CNDgccorrect=1                       # Run GC correction on read depth
CNDpileupMem=3000                    # Farm queue for CNDpileup [optional]
CNDpileupQueue=normal                # Memory usage for CNDpileup [optional]
CNDmem=2000                          # Farm queue for cnD [optional]
CNDqueue=normal                      # Memory usage for cnD [optional]

RDXplorer:
RDXout=NA18506.rdx.txt               # Output file name (in the /rdx/ dir)
RDXsplitBam=1                        # Create chrom. BAMs from a single BAM
RDXqueue=long                        # Farm queue for RDXplorer [optional]
RDXmem=2000                          # Memory usage for RDXplorer [optional]
```

SV caller score filtering:

```
BDscore=25                           # BreakDancerMax score cutoff
```

```
BDrs=2                                  # BreakDancerMax min. supporting RPs
PDscore=30                              # Pindel score cutoff
PDsupports=10                           # Pindel min. supporting reads
RDXscore=5                              # RDXplorer Z-score cutoff
```

Filtering by location:

```
bedexe=/path/to/BEDTools/bin/intersectBed   # BEDTools intersect binary
overlapexe=/path/to/SVMerge/overlapExons.pl # Included with SVMerge
gaps=/path/to//hg19_gap.txt                 # Sequence gaps to avoid
gapsBuffer=600                              # Distance from sequence gaps
centel=/path/to/hg19_cen_tel.txt            # Centromere and telomeres
centelBuffer=1000000                        # Distance from centromeres
gapOverlap=0.25                             # Fraction that must overlap gaps in
                                              order to be excluded
```

Contig assembly and alignments:

```
makeConfig=1                            # Create assembly config files from
                                          name.merged.tab
chrrefdir=/path/to/chromosome/dir/      # Path to directory with chromosome
                                          FASTA files (chromosome names must
                                          be the same as the BAM)
reffile=/path/to/ref/ref.fa             # FASTA file with reference chromosomes
submatrix=/path/to/SVMerge/submat.txt   # Provided with SVMerge package; used
                                          by Exonerate for alignment
joblimit=75                             # Max. assembly jobs to run from a
                                          job array [optional]
checkdone=1                             # Check for existing exonerate output
                                          and do not perform alignment if output
                                          file already exists
subseq=1                                # Align to a slice of the reference
                                          instead of a chromosome (use with
                                          offset=1)
outdir=.                                # Where to put the 'localAssemblies'
                                          directory if not in /project/version/
                                          svdir/
assemMin=1                              # run jobs 1-100 only
assemMax=100
assemQueue=normal                       # LSF/SGE job queue
assemMem=2000                           # LSF/SGE job memory requirement
```

Velvet parameters:

```
velvet=1                                # Set to 1 if velvet is used
velveth=/path/to/velveth                # Velveth binary
```

```
velvetg=/path/to/velvetg                # Velvetg binary
hashlen=29                              # Hash length (k-mer size)
ins_len=220                             # Library insert size
exp_cov=35                              # Expected coverage
cov_cutoff=2                            # Minimum coverage
```

ABySS parameters:

```
abyss=1                                 # Use ABySS for assembly
abyss-pe=/path/to/abyss-pe              # Path to ABySS-pe binary
kmer=25                                 # Hash length (k-mer size)
npairs=10                               # Minimum read pairs for scaffolding
```

Alignment parsing:

```
bamcheck=1                              # Apply read-depth check for deletions
meanCov=42                              # Expected mapped coverage (use with
                                          bamcheck=1)
zyg=het                                 # Zygosity of your genome (het or hom),
                                          use with bamcheck=1 for checking
                                          read-depth of deletions
offset=1                                # Apply offset (1 if subseq=1 was used)
parseJoblimit=10                        # Max. array jobs array to run simultaneously
parseSplit=1                            # Run splitFile.sh to split name.merged.tab
                                          for the alignment parsing step
parseSplitLines=250                     # Number of lines for each split
                                          name.merged.tab.* file
parseQueue=normal                       # LSF/SGE queue to submit parsing jobs
parseMem=2000                           # Memory usage for alignment parse jobs
```

References

SV callers:

Chen K, Wallis JW, McLellan MD, Larson DE, Kalicki JM, Pohl CS, McGrath SD, Wendl MC, Zhang Q, Locke DP et al: BreakDancer: an algorithm for high-resolution mapping of genomic structural variation. Nat Meth 2009, 6(9):677-681.

Simpson JT, McIntyre RE, Adams DJ, Durbin R: Copy number variant detection in inbred strains from short read sequence data. Bioinformatics, 26(4):565-567.

Ye K, Schulz MH, Long Q, Apweiler R, Ning Z: Pindel: a pattern growth approach to detect break points of large deletions and medium sized insertions from paired-end short reads. Bioinformatics 2009, 25(21):2865-2871.

Yoon S, Xuan Z, Makarov V, Ye K, Sebat J: Sensitive and accurate detection of copy number variants using read depth of coverage. Genome Research 2009, 19(9):1586-1592

Sequence assemblers:

Simpson JT, Wong K, Jackman SD, Schein JE, Jones SJM, Birol I: ABySS: A parallel assembler for short read sequence data. Genome Research 2009, 19(6):1117-1123.

Zerbino DR, Birney E: Velvet: Algorithms for *de novo* short read assembly using *de Bruijn* graphs. Genome Research 2008, 18(5):821-829.

Other tools:

Quinlan AR, Hall IM: BEDTools: a flexible suite of utilities for comparing genomic features. Bioinformatics, 26(6):841-842.

Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R: The Sequence Alignment/Map format and SAMtools. Bioinformatics 2009, 25(16):2078-2079.

Slater G, Birney E: Automated generation of heuristics for biological sequence comparison. BMC Bioinformatics 2005, 6(1):31.

`SVMerge`:

Wong K, Keane TM, Stalker J, Adams DJ: Enhanced structural variant and breakpoint detection using SVMerge by integration of multiple detection methods and local assembly. Genome Biol. 2010;11(12):R128. Epub 2010 Dec 31. http://genomebiology.com/content/11/12/R128

`BED` format:

http://genome.ucsc.edu/goldenPath/help/hgTracksHelp.html#BED