```
##############################################
#
# General description:
#
#   The following function reads a read list, a coverage and quantile matrix
#   (created by function CalcCoverMatReadList) determines insertion indicators
#   and saves results
#

# Input:
#
#     OutputFilePath: (character) path to file that will be written out
#     OutFolderName_NonRef: (character) path to folder that contains bam file
#         per peak
#     L1RangesPath: (character) path to file that contains results of comparing
#         peaks with L1 ranges (created by function ComparePeaksWithRefL1)

# Output:
#
#     ...

##############################################

AnalyzeReadMapped2L1 <- function(

  # Set parameters for range to determine start and end of 5' region of L1 stumps
  # and the minimum coverage in that region
  FivePStart = 50,
  FivePEnd   = 2000,
  MinCover   = 1,

  # Set parameter to determine whether a L1 insertion is potentially full-length
  # (it is not full-length if it has reads that are clipped by MinClip or more,
  # at position MaxL1Pos or less)
  MaxClip    = 100,
  MinL1End   = 6000,
  MaxL1Start = 100,

  # Set parameter for minimum proportion of a polymorphism among reads to be called
  MinPolyProp = 0.6,

  # Width of a motif to determine which side of a clipped read maps to L1
  MotifWidth  = 50,
  MotifOffSet = 60,

  # Specify flank size for sequences flanking L1 to be included in alignment output
  FlankSize = 2000,

  # Minimum number of bp clipped of a read for it to be included in alignment
  MinClipAlign = 1000,

  # Minimum length of full-length L1
  MinFullL1 = 6020,

  # Set file paths
  # Path to L1 GRanges from 1000 genome data
  CoveragePlotPath      = 'D:/L1polymORF/Figures/L1InsertionCoverage_NA12878_PacBio.pdf',
  CoverDataPath         = 'D:/L1polymORF/Data/L1_NA12878_PacBio_Coverage.RData',
  CoverDataPath         = 'D:/L1polymORF/Data/BZ_NA12878L1capt5-9kb_Results.Rdata',
  CoverDataPath         = 'D:/L1polymORF/Data/PacBioHiFi__Results.Rdata',
  PeakRangePath         = "D:/L1polymORF/Data/PacBioHiFiSubreads_L1Ranges.RData",
  RefRangePath          = "D:/L1polymORF/Data/L1RefRanges_hg19.RData",
  L1_1000GenomeDataPath = "D:/L1polymORF/Data/GRanges_L1_1000Genomes.RData",
  L1_1000_NA12878_Path  = "D:/L1polymORF/Data/NA12878.1000genome.L1HS.insert.bed",
  OutFolderName_NonRef  = "D:/L1polymORF/Data/BZ_NonRef",
  NewL1RefOutPath       = "D:/L1polymORF/Data/L1RefPacBioNA12878_DelRemoved.fa",
  GenomeBamPath         = "D:/L1polymORF/Data/BZ_NA12878L1capt5-9kb_subreads_hg19masked.sorted.bam",
  #GenomeBamPath         = "D:/L1polymORF/Data/BZ_NA12878L1capt5-9kb_subreads_hg19withL1.sorted.bam"
  ResultPath            = "D:/L1polymORF/Data/ReadsMapped2L1Info.RData",
  FastaFilePath         = "D:/L1polymORF/Data/",
  CatalogPath = "D:/L1polymORF/Data/L1Catalog_Updated_Wed_Aug_10_17-32-20_2016.csv"
){



################################
#                              #
#  Load and preprocess data    #
#                              #
################################
  L1InsertionFastaPath  <- paste(FastaFilePath, "L1InsertionWithFlank", FlankSize,
                                 "bp.fas", sep = "")


######
```

```
# Load ranges
######

# Load ranges
load(RefRangePath)
load(PeakRangePath)
load(L1_1000GenomeDataPath)

# Load coverage data (calculated in CalcCoverMatReadList)
load(CoverDataPath)

######
# Read and process L1 catalog
######

# Read in table with known L1
L1Catalogue <- read.csv(, as.is = T)

# Retain only entries with mapped L1 insertions and allele 1
blnL1Mapped       <- !is.na(L1Catalogue$start_HG38)
blnAllele1        <- L1Catalogue$Allele == 1
L1CatalogL1Mapped <- L1Catalogue[which(blnL1Mapped & blnAllele1),]

# Lift coordinates and get genomic ranges for catalog L1 on hg19
LiftOverList <- LiftoverL1Catalog(L1CatalogL1Mapped,
                                  ChainFilePath = "D:/L1polymORF/Data/hg38ToHg19.over.chain")
L1CatalogGR <- LiftOverList$GRCatalogue_hg19# Specify folder

######
# Read and process L1 consensus sequence
######

# Read in L1 consensus sequence
L1ConsSeq <- read.fasta("D:/L1polymORF/Data/Homo_sapiens_L1_consensus.fas")
L1ConsSeq <- toupper(L1ConsSeq[[1]])

# Turn into a DNAstring and create reverse complement
L1ConsSeqDNAst    <- DNAString(paste(L1ConsSeq, collapse = ""))
L1ConsSeqDNAst_RV <- reverseComplement(L1ConsSeqDNAst)
L1CharV           <- s2c(as.character(L1ConsSeqDNAst))
L1CharV_RV        <- s2c(as.character(L1ConsSeqDNAst_RV))

######
# Read and process repeatmasker table
######

# Read repeat table and subset to get only L1HS rows with fragment size below
# MaxFragLength
RepeatTable <- read.csv("D:/L1polymORF/Data/repeatsHg38_L1HS.csv")
#RepeatTable <- read.delim("D:/L1polymORF/Data/repeatL1hg19")
RepeatTable <- read.delim("D:/L1polymORF/Data/repeatsHg19_L1HS")
RepeatTable <- RepeatTable[RepeatTable$repName == "L1HS", ]

# Create genomic ranges for L1 fragments, match them to distances to get distance
# to consensus per fragment
L1RefGR <- GRanges(seqnames = RepeatTable$genoName,
                   ranges = IRanges(start = RepeatTable$genoStart,
                                    end = RepeatTable$genoEnd),
                   strand = RepeatTable$strand)
L1RefGRFull <- L1RefGR[width(L1RefGR) > 6000]

L1RefSeq <- getSeq(BSgenome.Hsapiens.UCSC.hg19, L1RefGRFull)
L1RefSeq <- as.character(L1RefSeq)
StrandV  <- as.vector(strand(L1RefGRFull))
L1RefSeqMat <- sapply(1:length(L1RefSeq), function(i) {
  L1V <- strsplit(L1RefSeq[i], "")[[1]]
  if (StrandV[i] == "+"){
    L1V[1:FivePEnd]
  } else {
    L1V[(length(L1V) - FivePEnd + 1):length(L1V)]
  }
})
dim(L1RefSeqMat)
colnames(L1RefSeqMat) <- paste(as.vector(seqnames(L1RefGRFull)), start(L1RefGRFull),
                               end(L1RefGRFull), "Ref", sep = "_")

##############################################
#                                            #
#  Collect info on promising L1 insertions   #
#            (FullL1Info)                    #
#                                            #
##############################################

# get names of newly created bam files
FileNames <- list.files(OutFolderName_NonRef, pattern = ".bam",
```

```
                        full.names = T)
FileNames <- FileNames[-grep(".bam.", FileNames)]

# Collect information on insertion that fullfill a certain minimum criterion
idx5P <- which(sapply(1:nrow(CoverMat), function(x) {
  all(CoverMat[x, FivePStart:FivePEnd] >= MinCover)
}))
cat("******  ", length(idx5P), "insertions have a minimum coverage of",
MinCover, "in 5' region from", FivePStart, "to", FivePEnd, "   *********\n")
idxFull <- which(sapply(1:nrow(CoverMat), function(x) all(CoverMat[x, FivePStart:6040] > 0)))
FullL1Info <- t(sapply(FilesWithReads[idx5P], function(x){
  FPathSplit <- strsplit(x, "/")[[1]]
  FName      <- FPathSplit[length(FPathSplit)]
  FName      <- substr(FName, 1, nchar(FName) - 4)
  strsplit(FName, "_")[[1]]
}))
FullL1Info <- base::as.data.frame(FullL1Info, stringsAsFactors = F)
colnames(FullL1Info) <- c("chromosome", "idx")
FullL1Info$Max3P <- sapply(idx5P, function(x) max(which(CoverMat[x, ] > 0)))
FullL1Info$idx   <- as.numeric(FullL1Info$idx)
FullL1Info$start <- start(IslGRanges_reduced[FullL1Info$idx])
FullL1Info$end   <- end(IslGRanges_reduced[FullL1Info$idx])
FullL1Info$cover   <- CoverMat[idx5P, 100]

idxFull2 <- which(rownames(FullL1Info) %in% FilesWithReads[idxFull])

# Set up insertion descriptors
FullL1GR <- makeGRangesFromDataFrame(FullL1Info)
FullL1Info$PotentialFullLength        <- NA
FullL1Info$L1InsertionPosition.median <- NA
FullL1Info$L1InsertionPosition.min    <- NA
FullL1Info$L1InsertionPosition.max    <- NA
FullL1Info$L1StrandNum                <- NA
FullL1Info$L1Strand                   <- NA
FullL1Info$L1Start.median             <- NA
FullL1Info$L1Start.min                <- NA
FullL1Info$L1Start.max                <- NA
FullL1Info$L1End.median               <- NA
FullL1Info$L1End.min                  <- NA
FullL1Info$L1End.max                  <- NA
FullL1Info$L15PTransdSeq.median       <- NA
FullL1Info$L15PTransdSeq.min          <- NA
FullL1Info$L15PTransdSeq.max          <- NA
FullL1Info$L13PTransdSeq.median       <- NA
FullL1Info$L13PTransdSeq.min          <- NA
FullL1Info$L13PTransdSeq.max          <- NA
FullL1Info$NrSupportReads             <- NA
FullL1Info$NrReadsCover5P             <- NA
FullL1Info$NrReadsCover3P             <- NA
FullL1Info$NrReadsReach5P             <- NA
FullL1Info$NrReadsReach3P             <- NA

# Generate a list of matched reads mapped to genome and to L1
MatchedReadList <- lapply(1:nrow(FullL1Info), function(i) {

  # Get position of current entry in ReadListPerPeak
  x <- idx5P[i]

  # Get reads mapped to L1, retain only primary reads and get the number of bp
  # clipped on the left
  RL          <- ReadListPerPeak[[x]]
  primMap     <- RL$flag <= 2047 & !(is.na(RL$pos))
  RL          <- lapply(RL, function(y) y[primMap])
  LRClippedL1 <- sapply(RL$cigar, NrClippedFromCigar, USE.NAMES = F)
  L1Length    <- width(RL$seq) - LRClippedL1[1,]

  # Get reads mapped to the genome on current locus
  ScanParam <- ScanBamParam(what = scanBamWhat(), which = FullL1GR[i])
  GenomeRL  <- scanBam(GenomeBamPath,  param = ScanParam)
  primMap     <- GenomeRL[[1]]$flag <= 2047 & !(is.na(GenomeRL[[1]]$pos))
  GenomeRL[[1]]    <- lapply(GenomeRL[[1]], function(y) y[primMap])
  LRClippedG <- sapply(RL$cigar, NrClippedFromCigar)
  ReadMatch <- match(RL$qname, GenomeRL[[1]]$qname)
  if (any(is.na(ReadMatch))){
    browser()
  }

  # Generate a list of reads mapped to genome, matching the list of reads
  # mapped to L1
  GenomeMatchRL <- lapply(GenomeRL[[1]], function(y) y[ReadMatch])
  list(L1ReadList = RL, GenomeReadList = GenomeMatchRL)
})
names(MatchedReadList) <- paste(FullL1Info$chromosome, FullL1Info$idx, sep = "_")

# Loop through list of matching reads and collect info
```

```
ReadInfoList <- lapply(MatchedReadList, function(RLL) {

  # Determine whether reads are on the same strand
  blnSameStrand <- RLL$GenomeReadList$strand == RLL$L1ReadList$strand

  # Loop through reads an collect information on number of bps clipped on
  # reads mapped to genome, the insertion location and the insertion strand
  ClipWithL1 <- 0
  L1Info <- t(sapply(1:length(RLL$GenomeReadList$pos), function(y) {

    # Get sequence and nr clipped of read mapped to L1
    L1Seq        <- RLL$L1ReadList$seq[y]
    LRClipped_L1 <- NrClippedFromCigar(RLL$L1ReadList$cigar[y])

    # Get start and end position on L1
    L1Start <- RLL$L1ReadList$pos[y]
    L1End   <- L1Start + ReadLengthFromCigar(RLL$L1ReadList$cigar[y])

    # Get sequence and nr clipped of read mapped to genome
    GSeq        <- RLL$GenomeReadList$seq[y]
    LRClipped_G <- NrClippedFromCigar(RLL$GenomeReadList$cigar[y])

    # Determine the clipped sequence of read mapped to genome
    LeftClippedSeq  <- subseq(GSeq, 1, LRClipped_G[1])
    RightClippedSeq <- subseq(GSeq, width(GSeq) - LRClipped_G[2], width(GSeq))

    # Get a part of read mapped to L1 and determine whether it is in the left
    # or right clipped sequence of read mapped to genome
    L1Motif         <- subseq(L1Seq, LRClipped_L1[1] + MotifOffSet,
                              LRClipped_L1[1] + MotifOffSet + MotifWidth)
    if (!blnSameStrand[y]) {L1Motif <- reverseComplement(L1Motif)}
    motifMatchLeft  <- matchPattern(L1Motif[[1]], LeftClippedSeq[[1]])
    motifMatchRight <- matchPattern(L1Motif[[1]], RightClippedSeq[[1]])
    blnL1OnLeft  <- length(motifMatchLeft)  > 0
    blnL1OnRight <- length(motifMatchRight) > 0

    # Identify position of L1 insertion
    InsPos <- NA
    if (blnL1OnLeft){
      InsPos <- RLL$GenomeReadList$pos[y]
    }
    if (blnL1OnRight){
      InsPos <- RLL$GenomeReadList$pos[y] +
        ReadLengthFromCigar(RLL$GenomeReadList$cigar[y])
    }
    # Scenario: 0 = L1 on negative strand and read maps on left side of L1
    #           1 = L1 on positive strand and read maps on left side of L1
    #           2 = L1 on negative strand and read maps on right side of L1
    #           3 = L1 on positive strand and read maps on right side of L1
    Scenario <- blnSameStrand[y] + 2 * blnL1OnLeft

    if (blnL1OnLeft & blnL1OnRight){
      warning("L1 motif matches left and right clipped sequence in", y, "\n", immediate. = T)
      Scenario <- NA
      ClipWithL1 <- NA
    }
    if ((length(motifMatchLeft) == 0) & (length(motifMatchRight) == 0)){
      #browser()
      warning("L1 motif matches no clipped sequence in read", y, "\n")
      Scenario <- NA
      ClipWithL1 <- NA
    }

    # Get start and stop indices of transduced sequence of 5' and 3' end
    TD5Pstart <- switch(as.character(Scenario),
                        '0' = width(GSeq) - LRClipped_L1[1],
                        '1' = width(GSeq) - LRClipped_G[2],
                        '2' = width(GSeq) - LRClipped_L1[1],
                        '3' = 1, NA)
    TD5Pend   <- switch(as.character(Scenario),
                        '0' = width(GSeq),
                        '1' = LRClipped_L1[1],
                        '2' = LRClipped_G[1],
                        '3' = LRClipped_L1[1])
    TD3Pstart <- switch(as.character(Scenario),
                        '0' = width(GSeq) - LRClipped_G[2],
                        '1' = width(GSeq) - LRClipped_L1[2],
                        '2' = 1,
                        '3' = width(GSeq) - LRClipped_L1[2], NA)
    TD3Pend <- switch(as.character(Scenario),
                      '0' = LRClipped_L1[2],
                      '1' = width(GSeq),
                      '2' = LRClipped_L1[2],
                      '3' = LRClipped_G[1], NA)
```

```
    # Determine which junction is covered by a read
    JunctionCovered <- switch(as.character(Scenario),
                        '0' = 3,
                        '1' = 5,
                        '2' = 5,
                        '3' = 3, NA)
    # Determine whether read reaches into 5' or 3' junction
    if (!is.na(ClipWithL1)){
      ClipWithL1 <- c(1:2)[c(blnL1OnLeft, blnL1OnRight)]
    }
    Reaches5P  <- JunctionCovered == 5 | LRClipped_G[ClipWithL1] > MinFullL1
    Reaches3P  <- JunctionCovered == 3 | LRClipped_G[ClipWithL1] > MinFullL1

    # Collect info in a vector
    c(LeftClipped_G = LRClipped_G[1], RightClipped_G = LRClipped_G[2],
      LeftClipped_L1 = LRClipped_L1[1], RightClipped_L1 = LRClipped_L1[2],
      InsPos = InsPos, Scenario = Scenario, TD5Pstart = TD5Pstart,
      TD5Pend = TD5Pend, TD5Pwidth = TD5Pend - TD5Pstart,
      TD3Pstart = TD3Pstart, TD3Pend = TD3Pend, TD3Pwidth = TD3Pend - TD3Pstart,
      JunctionCovered = JunctionCovered, L1Start = L1Start, L1End = L1End,
      Reaches5P = Reaches5P, Reaches3P = Reaches3P)
  }))
  L1Info <- as.data.frame(L1Info)
  L1Info$Name     <- RLL$GenomeReadList$qname
  L1Info$L1Strand <- 1*blnSameStrand # 0 corresponds to negative strand
  L1Info$L1pos    <- RLL$L1ReadList$pos
  L1Info$Scenario[is.na(L1Info$InsPos)] <- NA
  L1Info
})

# Loop through insertions and extract info
for (i in 1:length(ReadInfoList)){

  # Get Info
  L1Info   <- ReadInfoList[[i]]

  # Check whether any read has a long left-clipped pr right-clipped sequence
  # on L1 that indicates L1 insertion might not be full-length
  blnNotFull <- (L1Info$LeftClipped_L1 > MaxClip & L1Info$L1Start > MaxL1Start) |
                (L1Info$RightClipped_L1 > MaxClip & L1Info$L1End < MinL1End)
  FullL1Info$PotentialFullLength[i] <- !blnNotFull

  # Fill in info about L1 insertion (strandedness and position)
  FullL1Info$L1StrandNum[i] <- mean(L1Info$L1Strand, na.rm = T)
  if (mean(L1Info$L1Strand, na.rm = T) > 0.5) {
    FullL1Info$L1Strand[i] <- "+"
  } else {
    FullL1Info$L1Strand[i] <- "-"
  }
  FullL1Info$L1InsertionPosition.median[i] <- median(L1Info$InsPos)
  FullL1Info$L1InsertionPosition.min[i]    <- min(L1Info$InsPos)
  FullL1Info$L1InsertionPosition.max[i]    <- max(L1Info$InsPos)

  # Determine which reads cover which junction
  bln5Covered <- L1Info$JunctionCovered == 5
  bln3Covered <- L1Info$JunctionCovered == 3

  # Get start and end of L1
  FullL1Info$L1Start.median[i] <- median(L1Info$L1Start[L1Info$Reaches5P])
  FullL1Info$L1Start.min[i]    <- min(L1Info$L1Start[L1Info$Reaches5P])
  FullL1Info$L1Start.max[i]    <- max(L1Info$L1Start[L1Info$Reaches5P])
  FullL1Info$L1End.median[i]   <- median(L1Info$L1End[L1Info$Reaches3P])
  FullL1Info$L1End.min[i]      <- min(L1Info$L1End[L1Info$Reaches3P])
  FullL1Info$L1End.max[i]      <- max(L1Info$L1End[L1Info$Reaches3P])

  # Get length of 5' transduced sequence
  FullL1Info$L15PTransdSeq.median[i] <- median(L1Info$TD5Pwidth[bln5Covered])
  FullL1Info$L15PTransdSeq.min[i]    <- min(L1Info$TD5Pwidth[bln5Covered])
  FullL1Info$L15PTransdSeq.max[i]    <- max(L1Info$TD5Pwidth[bln5Covered])

  # Get length of 3' transduced sequence
  FullL1Info$L13PTransdSeq.median[i] <- median(L1Info$TD3Pwidth[bln3Covered])
  FullL1Info$L13PTransdSeq.min[i]    <- min(L1Info$TD3Pwidth[bln3Covered])
  FullL1Info$L13PTransdSeq.max[i]    <- max(L1Info$TD3Pwidth[bln3Covered])

  # Get number of supporting reads
  FullL1Info$NrReads5P[i] <- sum(L1Info$JunctionCovered == 5)
  FullL1Info$NrReads3P[i] <- sum(L1Info$JunctionCovered == 3)
  FullL1Info$NrReadsCover5P[i] <- sum(L1Info$JunctionCovered == 5)
  FullL1Info$NrReadsCover3P[i] <- sum(L1Info$JunctionCovered == 3)
  FullL1Info$NrReadsReach5P[i] <- sum(L1Info$Reaches5P)
  FullL1Info$NrReadsReach3P[i] <- sum(L1Info$Reaches3P)
  FullL1Info$NrSupportReads[i] <- nrow(L1Info)
}
cat("******  ", sum(FullL1Info$PotentialFullLength, na.rm = T),
```

```
    "insertions are potentially full length *********\n")

# Write out table with L1 info
write.csv(FullL1Info, file = "D:/L1polymORF/Data/L1InsertionInfo.csv")
}
```