Msc Internship Intermittent Project Proposal:

Variant Calling in R: Appendix

Dymphe Remarque,
S1838768

Supervisor:
Dr. Klaas Vrieling

Appendix:

1. Additional Graphs

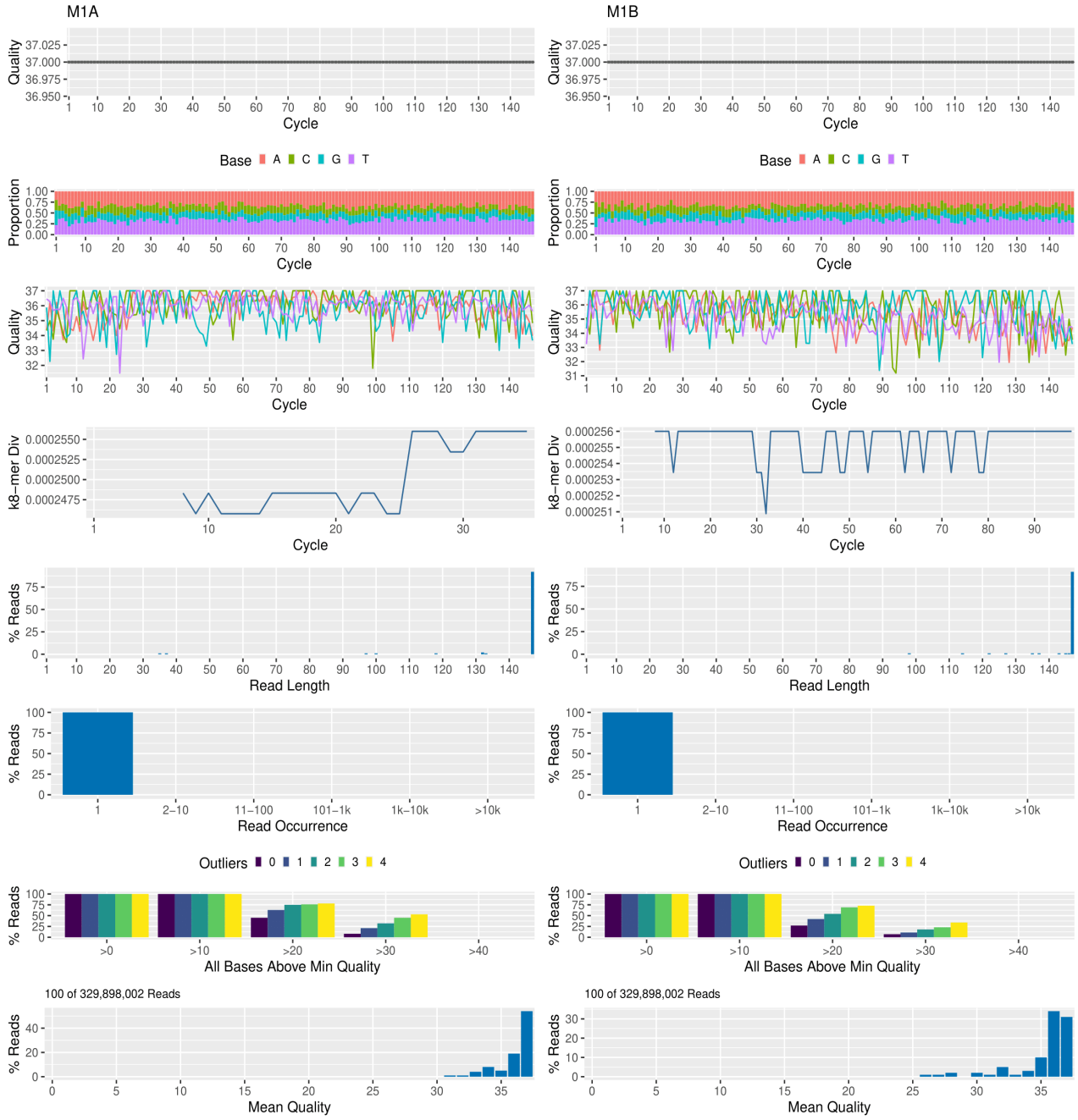

Figure 1: Graph of Citrix sequence data quality after filtering and adapter removal. Due to the sampling nature of the plots, the higher quality values may deviate from those of previous plots . Nevertheless, the figure indicates that quality filtering succeeded to exclude only the reads below the quality thresholds without affecting the nucleotide distribution, read complexity or nucleotide specific qualities negatively. Note how the k8-mer diversity now displays the diversity only until the end of the shortest sampled read due to the nature of the calculations. Note also how this shortest read length differs between the forward and reverse reads, which is caused by sampling individually rather than sampling read pairs. In this way, more quality information is gathered for the entire dataset with the downside of not being able to directly compare read pairs. This sampling method was chosen as the overall quality is more relevant for selecting filtering criteria than the comparison of quality between mates.

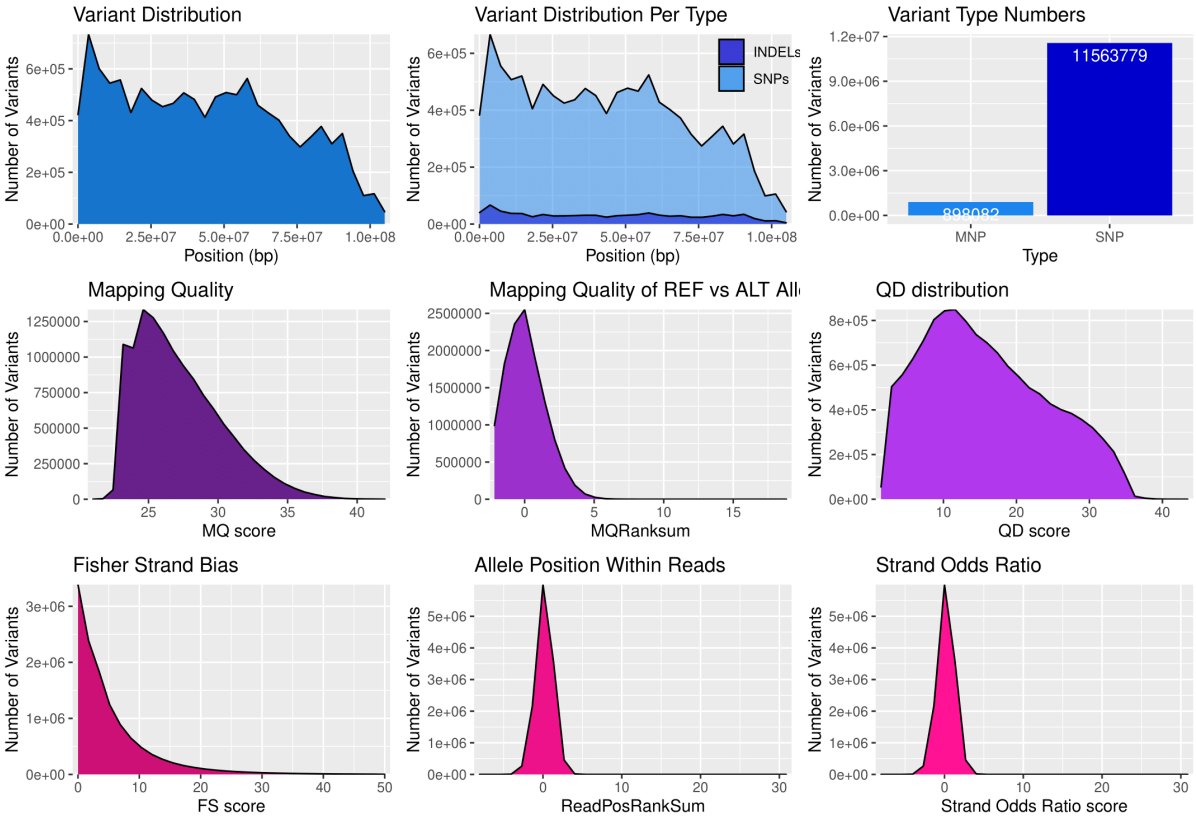

Figure 2: The graph of the selected high quality variants. Note how in comparison the the unfiltered quality graph, the variables have sharp drop-off where the thresholds were set. In addition, long ended tails such as that from the FS, ReadPosRankSum and SOR have been cut off, allowing the graph to zoom in on the large peaks around zero.
The majority of removed variants were microsatellites, as can be seen from the altered ratios in the top right graph. This is caused by the inherently lower confidence in variants with longer basepair lengths. For this reason, no strict thresholds were set that would cause removal of all INDELs.

2. Installation instructions additional software

Docker Toolbox

To access the BWA and GATK components, the Unix based software requires a virtual Linux environment provided by a docker. For this end, Docker Toolbox was implemented for Windows systems. To aid the user in preparing the virtual environment, installation instructions have been added here:

1. Download the Docker Toolbox installation software. For Windows, select the latest .exe file and for Mac, select the latest .pkg file.
2. Double click the downloaded file and follow the installation prompts
3. To verify the installation, run the Docker Quickstart terminal once. This will setup the virtual environment.
4. For Windows, check that the Docker Quickstart terminal installed Git Bash (bash.exe) at the default location: C:/Program Files/Git/bin/bash.exe
   For Mac, check that the terminal.exe is at its default location: /Applications/Utilities/Terminal.app.

Note that Docker Toolbox is not available for Linux systems as these do not require a virtual Linux environment. To access the GATK and BWA functions, the base software itself has to be installed. The latest installation software versions and instructions can be found here for GATK, BWA and Samtools and Picard (to support GATK).

3. Full function code and argument explanations

A. ***AlignQual()***

Based on RSamtools and ggplot2, this function is used collect quality data from an alignment BAM file, which is then reported in a mapQ graph, a table read numbers (un)mapped and ,optionally, coverage and alignment rates are outputted as well in the console.

Alignment_File_Name	The file name of the alignment to be assessed (including .bam file extension)
Reference_Length	The length of the utilised reference genome in basepairs. This is used to calculate the coverage based on the Lander/Waterman equation (Coverage=Length*Number of mapped reads/ Genome Length)
Read_Length	The general length of the sequenced reads in basepairs. This generally between 100 and 150 bp for Illumina.
Server	To save the quality plots on a server without gpu access, set this variable to TRUE to use an alternative save method.

Source Code

```
AlignQual<-function(Alignment_File_Name="No Default",
                    Reference_Length="No Default",
                    Read_Length="No Default",
                    server=F){

# Check Input
if(Alignment_File_Name=="No Default"){return(message("No alignment file has been
submitted"))
}else if(Reference_Length=="No Default" & Read_Length=="No Default"){
  message("Coverage and alignment rates will not be calculated due to missing
Reference_Length")}

# Fetch Mapping Qualities
  message("Collecting data...")
param<-Rsamtools::ScanBamParam(what=c("mapq"))
BAM_Frame<-as.data.frame(Rsamtools::scanBam(file=Alignment_File_Name ,param=param)
)

if(server==F){
# Plot Mapping Qualities
  message(" ")
  message("Plotting quality...")
Bam_Qual_Plot<-paste(gsub(".bam","", Alignment_File_Name),"_Align_Qual.pdf",sep=""
)
pdf(file=file.path(getwd(),Bam_Qual_Plot), width=11.7, height=5.7)
ggplot2::ggplot(data=BAM_Frame)+ggplot2::geom_density(ggplot2::aes(x=mapq),stat="b
in")
dev.off()
  message(" ")
  message("The Mapping Quality Graph has been saved in ",getwd()," as ",Bam_Qual_P
lot)
}else if(server==T){
  message(" ")
  message("Plotting quality...")
Bam_Qual_Plot<-paste(gsub(".bam","", Alignment_File_Name),"_Align_Qual.pdf",sep=""
```

```
)
bitmap(file=file.path(getwd(),Bam_Qual_Plot), width=11.7, height=5.7, type="pdfwri
te", res=100)
ggplot2::ggplot(data=BAM_Frame)+ggplot2::geom_density(ggplot2::aes(x=mapq),stat="b
in")
dev.off()
  message(" ")
  message("The Mapping Quality Graph has been saved in ",getwd()," as ",Bam_Qual_P
lot)
  }

# View Mapped/Unmapped ratios
  message(" ")
  message("Collecting BAM Statistics")
BAM_Summary<-capture.output(Rsamtools::quickBamFlagSummary(Alignment_File_Name))
Statistics<-paste(gsub(".bam","", Alignment_File_Name),"_Statistics.txt",sep="")
cat(BAM_Summary, file=Statistics,sep="\n")
  message(" ")
  message("The General_Alignment_Statistics have been saved as ", Statistics," in
", getwd())

# Calculate coverage and alignment rates and if possible
if(Reference_Length!="No Default" & Read_Length!="No Default"){
  stats<-read.delim(Statistics,stringsAsFactor=F)
  # For paired end data
  if(is.na(stats[25])){
    Coverage<-round((
      as.numeric(sub("\\D*(\\d+).*", "\\1", stats[3,]))-
        as.numeric(sub("\\D*(\\d+)\\D*(\\d+).*", "\\2", stats[20,]))-
        as.numeric(sub("\\D*(\\d+)\\D*(\\d+).*", "\\2", stats[25,])))*Read_Length
      /Reference_Length)
    Coverage<-paste(Coverage,"x",sep="")
    Rate<-round((as.numeric(sub("\\D*(\\d+).*", "\\1", stats[3,]))-
                   as.numeric(sub("\\D*(\\d+)\\D*(\\d+).*", "\\2", stats[20,]))-
                   as.numeric(sub("\\D*(\\d+)\\D*(\\d+).*", "\\2", stats[25,])))/
                  as.numeric(sub("\\D*(\\d+).*", "\\1", stats[3,]))*100,2)
    Rate<-paste(Rate,"%",sep="")
  }else{
  # For single end data
    Coverage<-round(
      (as.numeric(sub("\\D*(\\d+).*", "\\1", stats[3,]))-
        as.numeric(sub("\\D*(\\d+)\\D*(\\d+).*", "\\2", stats[20,]))-
        as.numeric(sub("\\D*(\\d+)\\D*(\\d+).*", "\\2", stats[25,])))*Read_Length/
Reference_Length)
    Coverage<-paste(Coverage,"x",sep="")
    Rate<-round((as.numeric(sub("\\D*(\\d+).*", "\\1", stats[3,]))-
                  as.numeric(sub("\\D*(\\d+)\\D*(\\d+).*", "\\2", stats[19,]))
                  /as.numeric(sub("\\D*(\\d+).*", "\\1", stats[3,]))*100),2)
    Rate<-paste(Rate,"%",sep="")
  }
  message("The coverage has been estimated to be ",Coverage, " with an alignment r
ate of ",Rate)
  message("")
}}
```

B. ***Bowtie2Align()***

Based on Rbowtie2 and RSamtools, this function is used to map both single and paired-end sequence data to a reference genome, followed by converting the SAM output into a sorted and indexed BAM file.

Reference_Genome	The reference genome unto which the sequence data will be mapped (including .fasta or .fna file extension)
Forward_Fastq	The single-read or forward paired-end fastq file name (including .fastq file extension)
Reverse_Fastq	When Paired_End_Data=TRUE, the reverse paired-end fastq file name (including .fastq file extension). This argument is ignored when Paired_End_Data=FALSE.
Paired_End_Data	The input data type, where FALSE corresponds with single-end reads. [default=TRUE]
Index_File_Name	The base name for the Bowtie2 reference index. If no index was made before, a new one will be constructed. Otherwise, the index with this basename will be used.
Output_Base_Name	The base name for the alignment files and reports. No file extension is required. [default=Bowtie2_Alignment]
Flag	Alignment data is supplied with data information such as read group identifiers (--rg-id), Sample name (--rg SM:), library (rg LB:), Predicted median insert size (--rg PI:) and platform (--rg PL:). [default="--rg-id 1 --rg SM:Sample --rg LB:1 --rg PI:400 --rg PL:ILLUMINA"]
Threads	The number of threads to use in alignment calculations. [default=2]

Source Code

```
Bowtie2Align<-function(Reference_Genome="No Default",
                       Forward_Fastq="No Default",
                       Reverse_Fastq="No Default",
                       Paired_End_Data=TRUE,
                       Index_File_Name="index",
                       Output_Base_Name="Bowtie2_Alignment",
                       Threads=4,
                       flag="--rg-id 1 --rg SM:Sample --rg LB:1 --rg PI:400 --rg
                             PL:ILLUMINA"){
 # Check Input
 if(Reference_Genome=="No Default"){return(message("No Reference genome has been
submitted"))
  }else if(file.exists(Forward_Fastq)==FALSE){return(message("No existing Fastq fi
les have been submitted"))
  }else if(file.exists(Reverse_Fastq)==FALSE & Paired_End_Data==TRUE){return(messa
ge("No existing reverse fastq file has been submitted, but paired_end_data=TRUE"))
  }else if(file.exists(paste(Output_Base_Name,".bam",sep=""))){return(message("The
submitted output files already exist in the working directory. Please choose a dif
ferent Output_Base_Name."))}

# Setup General Parameter Names
threads<-paste("--threads ",Threads)
threads2<-paste(flag, " --time --threads ",Threads)

# Alignment for Windows
if(.Platform$OS.type=="windows"){

# Build Index if required
if(!file.exists(paste(Index_File_Name,".1.bt2",sep=""))){
    message("Building Reference Genome index")
    message("")
```

```
  command<-paste(sep="",'"',.libPaths()[1],'/Rbowtie2/bowtie2-build-s.exe" -c ',fi
le.path(getwd(), Reference_Genome)," ",file.path(getwd(),Index_File_Name)," ",thre
ads)
  system(command, intern=TRUE)
}else{message("Previously made Reference Genome index located");message("")}

# Alignment for single end data
if(Paired_End_Data==FALSE){
  message("Aligning Reads. This will take a while")
  message("")
command<-paste(sep="",'"',.libPaths()[1],'/Rbowtie2/bowtie2-align-s.exe" -x ',file
.path(getwd(), Index_File_Name)," -r ",file.path(getwd(),Forward_Fastq)," -S ",fil
e.path(getwd(),Output_Base_Name)," ",threads2)
Log_Table_Alignment<-system(command, intern=TRUE)

}else{
#  Alignment for paired-end data
  message("Aligning Reads. This will take a while")
  message("")
command<-paste(sep="",'"',.libPaths()[1],'/Rbowtie2/bowtie2-align-s.exe" -x ',file
.path(getwd(), Index_File_Name)," -1 ",file.path(getwd(),Forward_Fastq)," -2 ",fil
e.path(getwd(),Reverse_Fastq) ," -S ",Output_Base_Name," ",threads2)
Log_Table_Alignment<-system(command, intern=TRUE)
}

}else{
#Alignment for Unix and Mac

# Build Index if required
if(!file.exists(paste(Index_File_Name,".1.bt2",sep=""))){
  file.create(Index_File_Name)
    message("Building Reference Genome index")
    message("")
  index<-Rbowtie2::bowtie2_build(references=Reference_Genome,
                                 bt2Index=file.path(getwd(),Index_File_Name),
                                 overwrite=TRUE,
                                 threads)
  }else{message("Previously made Reference Genome index located");message("")}

# Alignment for single end data
if(Paired_End_Data==FALSE){
  message("Aligning Reads. This will take a while")
  message("")
  Log_Table_Alignment<-Rbowtie2::bowtie2(bt2Index = file.path(getwd(), Index_File_
Name),
                                         samOutput = file.path(getwd(), Output_Bas
e_Name),
                                         seq1=Forward_Fastq,
                                         overwrite=TRUE,
                                         threads2)
}else{
# Alignment for paired-end data
  message("Aligning Reads. This will take a while")
  message("")
Log_Table_Alignment<-Rbowtie2::bowtie2(bt2Index = file.path(getwd(), Index_File_Na
me),
                                       samOutput = file.path(getwd(), Output_Base_
Name),
                                       seq1=Forward_Fastq,
```

```
                                        seq2=Reverse_Fastq,
                                        overwrite=TRUE,
                                        threads2)
}}
  message("The alignment has finished and has been saved as ", Output_Base_Name)
  message(" ")
# Store alignment rates in a special .txt file
  message("Storing alignment rates")
  message(" ")
Alignment_File_NameX<-paste(gsub(".sam","", Output_Base_Name),"_Alignment_Rates.tx
t",sep="")
cat(Log_Table_Alignment, file=Alignment_File_NameX,sep="\n")
  message("Aligment rates have been stored as ",Alignment_File_NameX)
  message("")

  message("Converting and processing output...")
# Convert the SAM File to a BAM File
  message("Converting Bam to Sam")
  message("")
BAM_File_Alignment<-Rsamtools::asBam(Output_Base_Name)

# Sort the BAM File
  message("Sorting Bam by coordinate")
  message("")
sortname<-gsub(".bam","", Output_Base_Name)
Rsamtools::sortBam(BAM_File_Alignment,sortname)

# Index the BAM File
  message("Indexing BAM file")
Rsamtools::indexBam(BAM_File_Alignment)

# Final Messages
  message("The entire alignment has been performed succesfully!")
  message("The Alignment Rates Are:")
print(Log_Table_Alignment)
  message(" ")
  message("The output file ",BAM_File_Alignment," and the corresponding .BAM index
file can be found at ",getwd())
}
```

C. ***BWAMEM_Align()***

This function allows the user to send commands to BWA-MEM, running either natively on linux and Mac or within a docker for Windows, from within R.

Reference_Genome	The reference genome unto which the sequence data will be mapped (including .fasta or .fna file extension)
Forward_Fastq	The single-read or forward paired-end fastq file name (including .fastq file extension)
Reverse_Fastq	When Paired_End_Data=TRUE, the reverse paired-end fastq file name (including .fastq file extension). This argument is ignored when Paired_End_Data=FALSE.
Paired_End_Data	The input data type, where FALSE corresponds with single-end reads. [default=TRUE]
Output_Base_Name	The base name for the alignment files and reports. No file extension is required. [default=Bowtie2_Alignment]

Flag	Alignment data is supplied with data information such as read group identifiers (--rg-id), Sample name (--rg SM:), library (rg LB:), Predicted median insert size (--rg PI:) and platform (--rg PL:). [default="--rg-id 1 --rg SM:Sample --rg LB:1 --rg PI:400 --rg PL:ILLUMINA"]
pathtoBWA	Only for running BWA on linux: the file path to the BWA software when a custom installation path has been used. The default is bwa, which assumes gatk is added to your $PATH.
pathtoGATK	Only for running GATK on linux: the file path to the GATK software when a custom installation path has been used. The default is ./gatk, which assumes gatk is added to your $PATH.
pathtobash	Only for running GATK within a docker (Mainly for windows users): the file path to git bash (bash.exe) when installed in a custom file location. Linux users can alter this directory path to make use of GATK within the docker environment as well.

Source Code

```
BWAMEM_Align<-function(Reference_Genome="No Default", Forward_Fastq="No Default
", Reverse_Fastq="No Default", Paired_End_Data=TRUE, Output_Base_Name="Bowtie2_
Alignment", pathtobash="C:/Program Files/Git/bin/bash.exe",pathtoBWA="bwa",pa
thtoGATK="./gatk", flag="--rg-id 1 --rg SM:Sample --rg LB:1 --rg PI:400 --rg
PL:ILLUMINA"){
  # Check Input
  if(Reference_Genome=="No Default"){return(message("No Reference genome has b
een submitted"))
  }else if(file.exists(Forward_Fastq)==FALSE){return(message("No existing Fastq
files have been submitted"))
  }else if(file.exists(Reverse_Fastq)==FALSE & Paired_End_Data==TRUE){return(mess
age("No existing reverse fastq file has been submitted, but paired_end_dat
a=TRUE"))
  }else if(file.exists(paste(Output_Base_Name,".bam",sep=""))){return(message("T
he submitted output files already exist in the working directory. Please c
hoose a different Output_Base_Name."))
  }else if(.Platform$OS.type=="windows" & file.exists("C:/Program Files/Git/bi
n/bash.exe")==FALSE){return(message("Git bash could not be found at the defi
ned location. Change pathtobash to the correct location or install git bas
h and the docker from https://docs.docker.com/toolbox/toolbox_install_wind
ows/ or https://docs.docker.com/engine/install/#server"))
  }

  # Variant Calling Windows version
  if(.Platform$OS.type=="windows" | pathtobash!="C:/Program Files/Git/bin/bash
.exe"){
    message("Using a docker environment")

    # Adapt windows file path to Unix commands
    if(.Platform$OS.type=="windows"){
      wd<-paste("//",gsub(":/","/",getwd()),sep="")
    }else{wd<-getwd()}

    # Add the file paths in front of input names
    Reference_GenomeP<-paste(wd,"/",Reference_Genome,sep="")
    Output_Base_NameS<-paste(Output_Base_Name,".sam",sep="")
    Output_Base_NameP<-paste(wd,"/",Output_Base_NameS,sep="")
```

```
    BAM_File_Alignment<-paste(Output_Base_Name,".bam",sep="")

    # Start docker
    message("Calling upon docker")
    shell(cmd = "'docker-machine restart'", shell=pathtobash, intern=T, flag =
"-c")

    # Check software installation
    message("Check for BWA and GATK software")
    shell(cmd = "'docker pull biocontainers/bwa:v0.7.17-3-deb_cv1
          docker pull broadinstitute/gatk:4.1.3.0'", shell=pathtobash, inter
n=T, flag = "-c")

    # Prepare the reference genome if this has not been done
    if(file.exists(paste(gsub("\\..*","",Reference_Genome),".fai",sep=""))==FALS
E){
      # Index the fasta file
      message("Indexing reference fasta file")
      command<-paste("'docker run -v ",wd,"://c/Users/Public biocontainers/b
wa:v0.7.17-3-deb_cv1 bwa index -a bwtsw -p ",Reference_GenomeP," -b 10000000
0'",sep="")
      shell(cmd = command, shell=pathtobash, intern=T, flag = "-c")

      if(file.exists(paste(gsub("\\..*","",Reference_Genome),".fai",sep=""))){
        message("The reference index has been made successfully")
      }else{message("No index has been made. Something went wrong, please vi
ew the samtools output above for details")}}

    # BWA Alignment
    if(Paired_End_Data==FALSE){
      message("Aligning Reads. This will take a while")
      message("")
      command<-paste("'docker run -v ",wd,"://c/Users/Public biocontainers/b
wa:v0.7.17-3-deb_cv1 bwa mem -M -R '@RG\tID:MK\tLB:MK\tPL:ILLUMINA\tSM:MK'
-t 4 ", Reference_GenomeP," ", Filtered_File_Name," -o //c/Users/Public/", Out
put_Base_NameS,"'",sep="")
      shell(cmd = command, shell=pathtobash, intern=T, flag = "-c")
    }else{
      message("Aligning Reads. This will take a while")
      message("")
      command<-paste("'docker run -v ",wd,"://c/Users/Public biocontainers/b
wa:v0.7.17-3-deb_cv1 bwa mem -M -R '@RG\tID:MK\tLB:MK\tPL:ILLUMINA\tSM:MK'
-t 4 ", Reference_GenomeP, Forward_Fastq," ", Reverse_Fastq," -o //c/Users/Publ
ic/", Output_Base_NameS,"'",sep="")
      shell(cmd = command, shell=pathtobash, intern=T, flag = "-c")
    }

    # Sortsam
    message("Sortting SAM by coordinate")
    message("")
    command<-paste("'docker run -v ",wd,"://c/Users/Public broadinstitute/ga
tk:4.1.3.0 ./gatk SortSam -I ",Output_Base_NameP," -O //c/Users/Public/",Out
put_Base_NameS," -SO coordinate'",sep="")
    shell(cmd = command, shell=pathtobash, intern=T, flag = "-c")
```

```
    # Mark Duplicates
    message("Marking duplicates")
    message("")
    command<-paste("'docker run -v ",wd,"://c/Users/Public broadinstitute/ga
tk:4.1.3.0 ./gatk MarkDuplicates -I ",Output_Base_NameP," -O //c/Users/Publi
c/",BAM_File_Alignment," -SO coordinate'",sep="")
    shell(cmd = command, shell=pathtobash, intern=T, flag = "-c")

    # Shut down docker
    message("The alignment and sorting steps have finished. Shutting down..
.")
    shell(cmd = "'docker-machine stop'", shell=pathtobash, intern=T, flag = "-c
")

  }else{
    # Variant Calling Unix and Mac version
    message("Using native software")
    # Add the file paths in front of input names
    wd<-paste("//",gsub(":/","/",getwd()),sep="")
    Reference_GenomeP<-paste(wd,"/",Reference_Genome,sep="")
    Output_Base_NameS<-paste(Output_Base_Name,".sam",sep="")
    Output_Base_NameP<-paste(wd,"/",Output_Base_NameS,sep="")
    BAM_File_Alignment<-paste(Output_Base_Name,".bam",sep="")

    # Prepare the reference genome if this has not yet been done
    if(file.exists(paste(gsub("\\..*","",Reference_Genome),".fai",sep=""))==FALS
E){
      # Index the fasta file
      message("Indexing reference fasta file")
      command<-paste(pathtoBWA," index -a bwtsw -p ",Reference_GenomeP," -b 100
000000'",sep="")
      system(command, intern=TRUE)

      if(file.exists(paste(gsub("\\..*","",Reference_Genome),".fai",sep=""))){
        message("The reference index has been made successfully")
      }else{message("No index has been made. Something went wrong, please vi
ew the samtools output above for details")}}

    # BWA Alignment
    if(Paired_End_Data==FALSE){
      message("Aligning Reads. This will take a while")
      message("")
      command<-paste(pathtoBWA," mem -M -R '@RG\tID:MK\tLB:MK\tPL:ILLUMINA\tSM
:MK' -t 4 ", Reference_GenomeP," ", Forward_Fastq," ", getwd(), "/", Output_Base
_NameS,sep="")
      system(command, intern=TRUE)
    }else{
      message("Aligning Reads. This will take a while")
      message("")
      command<-paste(pathtoBWA," mem -M -R '@RG\tID:MK\tLB:MK\tPL:ILLUMINA\tSM
:MK' -t 4 ", Reference_GenomeP, Forward_Fastq," ", Reverse_Fastq," ", getwd(), "
/", Output_Base_NameS,sep="")
      system(command, intern=TRUE)
    }
```

```
    # Sortsam
    message("Sorting SAM by coordinate")
    message("")
    command<-paste(pathtoGATK," SortSam -I ",Output_Base_NameP," ", getwd(), "/"
, Output_Base_NameS," -SO coordinate",sep="")
    system(command, intern=TRUE)

    # Mark Duplicates
    message("Marking duplicates")
    message("")
    command<-paste(pathtoGATK," MarkDuplicates -I ",Output_Base_NameP," ", getw
d(), "/", BAM_File_Alignment," -SO coordinate",sep="")
    system(command, intern=TRUE)
    message("The alignment has finished. The files can be found in ",getwd()
)}}
```

D. FastqQual()

Based on ShortRead, SystempipeR and Rbowtie2, this function allows you to create quality plots and potential adapter sequences of both single and paired-end sequence data.

Forward_Fastq	The single-read or forward paired-end fastq file name (including .fastq file extension)
Reverse_Fastq	When Paired_End_Data=TRUE, the reverse paired-end fastq file name (including .fastq file extension). This argument is ignored when Paired_End_Data=FALSE.
Paired_End_Data	The input data type, where FALSE corresponds with single-end reads. [default=TRUE]
Create_Graph	Whether to create a quality graph when Paired-End_Data=TRUE. [default=TRUE]
Adapter_Check	Whether to check for the presence of adapters when Paired-End_Data=TRUE. [default=TRUE]
Output_Base_Name	The base name of the generated quality plots. No file extension is required. [default="FastqQual"]
server	To save the quality plots on a server without gpu access, set this variable to TRUE to use an alternative save method.

Source Code

```
FastqQual<-function(Forward_Fastq="No Default", Reverse_Fastq="No Default", Paired
_End_Data=TRUE, Adapter_Check=TRUE, Create_Graph=TRUE, Output_Base_Name="FastqQual
",server=F){

# Check Input
if(Forward_Fastq=="No Default"){return(message("No Fastq file has been submitted")
)
}else if(Reverse_Fastq=="No Default" & Paired_End_Data==TRUE){return(message("No r
everse fastq file has been submitted, but paired_end_data=TRUE"))
}else if(Paired_End_Data==TRUE & Adapter_Check==FALSE & Create_Graph==FALSE){retur
n(message("Both adapter search and graph creation have been turned off. No analysi
s will be performed."))}

# Create Single Read Quality Plots
if(Paired_End_Data==FALSE){
  message("Generating Single Read Quality Plots. This may take a while")
```

```
  message("")
QA_Graph<-ShortRead::report(ShortRead::qa(Forward_Fastq,type="fastq"))
QA_Graph<-gsub("/index.html","",QA_Graph)
file.rename(from=QA_Graph, to=file.path(getwd(),Output_Base_Name))
 message("Quality plots have been saved in ", getwd()," in the ", Output_Base_Name
, " directory")

}else{

# Create Paired End Read Quality Plots
if(Create_Graph==TRUE){
# Make a named character vector and fetch relevant information
  message("Collecting data. This may take a while")
  message("")
faq_list<-c(file.path(getwd(),Forward_Fastq),file.path(getwd(),Reverse_Fastq))
names(faq_list)<-c("M1A","M1B")
fqlist<-systemPipeR::seeFastq(fastq=faq_list, batchsize = 100)

if(server==F){
# Plot the relevant information
  message("Generating Quality Plots. This may take a while")
  message("")
Output_Base_NameX<-paste(Output_Base_Name,".pdf",sep="")
pdf(file=file.path(getwd(),Output_Base_NameX), width=11.7, height=11.7)
systemPipeR::seeFastqPlot(fqlist)
dev.off()
rm(faq_list,fqlist)
  message("Quality plots have been saved in ", getwd(), " as ", Output_Base_NameX)
  message("")
}else if(server==T){
  message("Generating Quality Plots. This may take a while")
  message("")
Output_Base_NameX<-paste(Output_Base_Name,".pdf",sep="")
bitmap(file=file.path(getwd(),Output_Base_NameX), width=11.7, height=11.7, type="p
dfwrite", res=100)
systemPipeR::seeFastqPlot(fqlist)
dev.off()
rm(faq_list,fqlist)
  message("Quality plots have been saved in ", getwd(), " as ", Output_Base_NameX)
  message("")

  }
}

# Find Adapters in Unix and Mac OS
if(Adapter_Check==TRUE & .Platform$OS.type!="windows"){
  message("Searching for potential adapter sequences")
Output_Base_NameX<-paste(Output_Base_Name,".adapter1",sep="")
file.create(Output_Base_NameX)
Output_Base_NameX<-paste(Output_Base_Name,".adapter2",sep="")
file.create(Output_Base_NameX)
Adapters<-suppressWarnings(Rbowtie2::identify_adapters(file1=Forward_Fastq,
                                                       file2=Reverse_Fastq,
                                                       basename=file.path(getwd(),
Output_Base_Name),
                                                       overwrite=TRUE))

# Save Info in a .txt file (optional)
Adapter<-as.data.frame(Adapters)
```

```
Output_Base_NameX<-paste(Output_Base_Name,"_Adapters.txt",sep="")
row.names(Adapter)<-c("Forward Adapter", "Reverse Adapter")
write.table(Adapter,Output_Base_NameX,sep="\t")
  message("Adapter sequences have been saved in ",Output_Base_NameX)
}else if(Adapter_Check==TRUE & .Platform$OS.type=="windows"){

# Find Adapters in Windows OS
  message("Searching for potential adapter sequences")
Output_Base_NameX<-paste(Output_Base_Name,".adapter1",sep="")
file.create(Output_Base_NameX)
Output_Base_NameX<-paste(Output_Base_Name,".adapter2",sep="")
file.create(Output_Base_NameX)

command<-paste(sep="","'",.libPaths()[1],"/Rbowtie2/AdapterRemoval' --identify-ada
pters --file1 ",Forward_Fastq," --file2 ",Reverse_Fastq," --basename ",Output_Base
_Name)
Adapters<-system(command, intern=TRUE)

# Save Info in a .txt file (optional)
Adapter<-as.data.frame(Adapters)
Output_Base_NameX<-paste(Output_Base_Name,"_Adapters.txt",sep="")
row.names(Adapter)<-c("Forward Adapter", "Reverse Adapter")
write.table(Adapter,Output_Base_NameX,sep="\t")
  message("Adapter sequences have been saved in ",Output_Base_NameX)
}

# Clean up temporary files
Output_Base_NameX<-paste(Output_Base_Name,".adapter1",sep="")
file.remove(Output_Base_NameX)
Output_Base_NameX<-paste(Output_Base_Name,".adapter2",sep="")
file.remove(Output_Base_NameX)
}}
```

E. FastqTrim()

Based on Dada2 and Rbowtie2, this function allows you to filter both single and paired-end sequence data based on quality, as well as trimming of potential adapters.

Forward_Fastq	The single-read or forward paired-end fastq file name (including .fastq file extension)
Reverse_Fastq	When Paired_End_Data=TRUE, the reverse paired-end fastq file name (including .fastq file extension). This argument is ignored when Paired_End_Data=FALSE.
Paired_End_Data	The input data type, where FALSE corresponds with single-end reads. [default=TRUE]
Adapter_Removal	Whether to remove adapters sequences defined with Adapters="example.txt" or "sequence". [default=FALSE]
Adapters	Either an R character string object of forward and reverse adapters or the .txt file of the adapters generated by FastqQual()
Output_Base_Name	The base name of the generated trimmed fastq files. No file extension is required. [default="Trimmed_Fastq"]
server	To save the quality plots on a server without gpu access, set this variable to TRUE to use an alternative save method.
minLen	(Optional) The minimum read length allowed. [default=20]
maxLen	(Optional) The maximum read length allowed. [default=Inf, no limit]

maxN	(Optional) The maximum number of unknown nucleotides N allowed within a single read. [default=30]
minQ	(Optional) The minimum base quality score allowed. Reads with one or more bases below this threshold will be discarded. [default=10]
rm.phix	(Optional) Whether to remove common contraminator sequences recorded in the phiX genome [default=TRUE]
truncQ	(Optional) The minimum quality score of a base after which a read is truncated [default=2]
trimLeft	(Optional) The number of nucleotides to remove on the 3' end of all reads. Use this if the first sequence cycles are of low quality. [default=0]
trimRight	(Optional) The number of nucleotides to remove on the 5' end of all reads. Use this if the last sequence cycles are of low quality. [default=0]
maxEE	(Optional) The maximum expected error allowed for a single read. EE=sum(10^(-Q/10)). [default=Inf, No maximum]
rm.lowcomplex	(Optional) The minimum read complexity allowed, based on Shannon Entropy. [default=0]

Source Code

```
FastqTrim<-function(Forward_Fastq="No Default",
                    Reverse_Fastq="No Default",
                    Paired_End_Data=TRUE,
                    Adapter_Removal=FALSE,
                    Adapters="No Default",
                    Output_Base_Name="Trimmed_Fastq",
                    minLen=10, maxLen=Inf, maxN=30, minQ=10,
                    rm.phix=TRUE, truncQ=2, trimLeft=0, trimRight=0,
                    maxEE=Inf, rm.lowcomplex=0){
# Check Input
if(file.exists(Forward_Fastq)==FALSE){return(message("No existing Fastq file has b
een submitted"))
}else if(file.exists(Reverse_Fastq)==FALSE & Paired_End_Data==TRUE){return(message
("No existing reverse fastq file has been submitted, but paired_end_data=TRUE"))
}else if(Adapter_Removal==TRUE & Adapters=="No Default"){return(message("Adapter_R
emoval is set to true, but no adapter sequences have been provided. If adapters ar
e unknown, perform a search with FastqQual first."))
}

if(Paired_End_Data==FALSE){
## For Single Reads

if(Adapter_Removal==TRUE){
  if(grepl(".txt",Adapters)==1 & .Platform$OS.type!="windows"){
    Adapters<-as.character(read.table(Adapters)[,1])
    Temp_Filtered_File_Name<-paste(Output_Base_Name,"_temp.fastq",sep="")
       message("Removing adapters specified in .txt file")
    Rbowtie2::remove_adapters(file1=Forward_Fastq,
                              adapter1 = Adapters[1],
                              output1=file.path(getwd(),Temp_Filtered_File_Name),
                              overwrite=TRUE,
                              "--threads 2")
  }else if(grepl(".txt",Adapters)!=1 & .Platform$OS.type!="windows"){
    Temp_Filtered_File_Name<-paste(Output_Base_Name,"_temp.fastq",sep="")
       message("Removing adapters specified in R character strings")
    Rbowtie2::remove_adapters(file1=Forward_Fastq,
```

```
                                adapter1 = Adapters[1],
                                output1=file.path(getwd(),Temp_Filtered_File_Name),
                                overwrite=TRUE,
                                "--threads 2")
  }else if(grepl(".txt",Adapters)==1 & .Platform$OS.type=="windows"){
    Adapters<-as.character(read.table(Adapters)[,1])
    Temp_Filtered_File_Name<-paste(Output_Base_Name,"_temp.fastq",sep="")
       message("Removing adapters specified in R character strings")
    command<-paste(sep="","'",.libPaths()[1],"/Rbowtie2/AdapterRemoval' --file1 ",
Forward_Fastq," --adapter1 ",Adapters[1]," --output1 ",file.path(getwd(),Temp_Filt
ered_File_Name))
    system(command, intern=TRUE)
  }
  # Filter and Trim after adapter removal
    message("Filtering FastQ Files. This may take a while")
  Filtered_File_Name<-paste(Output_Base_Name,".fastq",sep="")
  dada2::filterAndTrim(fwd=Temp_Filtered_File_Name, filt=Filtered_File_Name,
                       truncQ=truncQ,
                       maxLen=maxLen,
                       minLen=minLen,
                       maxN=maxN,
                       minQ=minQ,
                       rm.phix=rm.phix,
                       truncLen=truncLen,
                       trimLeft=trimLeft,
                       trimRight=trimRight,
                       maxEE=maxEE,
                       rm.lowcomplex=rm.lowcomplex,
                       compress=FALSE, verbose=TRUE)
  message(" ")
  message("The Filtered FASTQ File has been saved in ", getwd()," as ", Filtered_F
ile_Name)
  file.remove(Temp_Filtered_File_Name)
}else{

# Filter and Trim without adapter removal
  message("Filtering FastQ Files. This may take a while")
Filtered_File_Name<-paste(Output_Base_Name,".fastq",sep="")
dada2::filterAndTrim(fwd=Forward_Fastq, filt=Filtered_File_Name,
                     truncQ=truncQ,
                     maxLen=maxLen,
                     minLen=minLen,
                     maxN=maxN,
                     minQ=minQ,
                     rm.phix=rm.phix,
                     truncLen=truncLen,
                     trimLeft=trimLeft,
                     trimRight=trimRight,
                     maxEE=maxEE,
                     rm.lowcomplex=rm.lowcomplex,
                     compress=FALSE, verbose=TRUE)
  message(" ")
  message("The Filtered FASTQ File has been saved in ", getwd()," as ", Filtered_F
ile_Name)
}
} else{
## Paired End reads
Forward_Filtered_File_Name<-paste(Output_Base_Name,"_fw.fastq",sep="")
Reverse_Filtered_File_Name<-paste(Output_Base_Name,"_rv.fastq",sep="")
```

```
# Remove Adapters
if(Adapter_Removal==TRUE){
  if(grepl(".txt",Adapters)==1 & .Platform$OS.type!="windows"){
Adapters<-as.character(read.table(Adapters)[,1])
Temp_FW_Filtered_File_Name<-paste(Output_Base_Name,"temp_fw.fastq",sep="")
Temp_RV_Filtered_File_Name<-paste(Output_Base_Name,"temp_rv.fastq",sep="")
  message("Removing adapters specified in .txt file")
Rbowtie2::remove_adapters(file1=Forward_Fastq,
                          file2=Reverse_Fastq,
                          adapter1 = Adapters[1],
                          adapter2 = Adapters[2],
                          output1=file.path(getwd(),Temp_FW_Filtered_File_Name),
                          output2=file.path(getwd(),Temp_RV_Filtered_File_Name),
                          overwrite=TRUE,
                          "--threads 2")
  }else if(grepl(".txt",Adapters)!=1 & .Platform$OS.type!="windows"){
    Temp_FW_Filtered_File_Name<-paste(Output_Base_Name,"temp_fw.fastq",sep="")
    Temp_RV_Filtered_File_Name<-paste(Output_Base_Name,"temp_rv.fastq",sep="")
      message("Removing adapters specified in R character strings")
    Rbowtie2::remove_adapters(file1=Forward_Fastq,
                              file2=Reverse_Fastq,
                              adapter1 = Adapters[1],
                              adapter2 = Adapters[2],
                            output1=file.path(getwd(),Temp_FW_Filtered_File_Name),
                            output2=file.path(getwd(),Temp_RV_Filtered_File_Name),
                              overwrite=TRUE,
                              "--threads 2")
  }else if(grepl(".txt",Adapters)==1 & .Platform$OS.type=="windows"){
    Adapters<-as.character(read.table(Adapters)[,1])
    Temp_FW_Filtered_File_Name<-paste(Output_Base_Name,"temp_fw.fastq",sep="")
    Temp_RV_Filtered_File_Name<-paste(Output_Base_Name,"temp_rv.fastq",sep="")
      message("Removing adapters specified in R character strings")

    command<-paste(sep="","'",.libPaths()[1],"/Rbowtie2/AdapterRemoval' --file1 ",
Forward_Fastq," --file2 ",Reverse_Fastq," --adapter1 ",Adapters[1]," --adapter2 ",
Adapters[2]," --output1 ",file.path(getwd(),Temp_FW_Filtered_File_Name)," --output
2 ",file.path(getwd(),Temp_RV_Filtered_File_Name))
    system(command, intern=TRUE)

  }else{
    Temp_FW_Filtered_File_Name<-paste(Output_Base_Name,"temp_fw.fastq",sep="")
    Temp_RV_Filtered_File_Name<-paste(Output_Base_Name,"temp_rv.fastq",sep="")
      message("Removing adapters specified in R character strings")

    command<-paste(sep="","'",.libPaths()[1],"/Rbowtie2/AdapterRemoval' --file1 ",
Forward_Fastq," --file2 ",Reverse_Fastq," --adapter1 ",Adapters[1]," --adapter2 ",
Adapters[2]," --output1 ",file.path(getwd(),Temp_FW_Filtered_File_Name)," --output
2 ",file.path(getwd(),Temp_RV_Filtered_File_Name))
      system(command, intern=TRUE)
  }
# Filter and Trim after adapter removal
  message("Filtering FastQ Files. This may take a while")
dada2::filterAndTrim(fwd=Temp_FW_Filtered_File_Name,
                     filt=Forward_Filtered_File_Name,
                     rev=Temp_RV_Filtered_File_Name,
                     filt.rev=Reverse_Filtered_File_Name,
                     truncQ=truncQ,
                     maxLen=maxLen,
```

```
                      minLen=minLen,
                      maxN=maxN,
                      minQ=minQ,
                      rm.phix=rm.phix,
                      truncLen=truncLen,
                      trimLeft=trimLeft,
                      trimRight=trimRight,
                      maxEE=maxEE,
                      rm.lowcomplex=rm.lowcomplex,
                      compress=FALSE, verbose=TRUE)
file.remove(Temp_FW_Filtered_File_Name)
file.remove(Temp_RV_Filtered_File_Name)
  message(" ")
  message("The Filtered FASTQ Files have been saved in ", getwd()," as",   Forward
_Filtered_File_Name, " and ",Reverse_Filtered_File_Name)
}else{
# Filter and Trim without adapter removal
  message("Filtering FastQ Files. This may take a while")
dada2::filterAndTrim(fwd=Forward_Fastq, filt=Forward_Filtered_File_Name,
                      rev=Reverse_Fastq, filt.rev=Reverse_Filtered_File_Name,
                      truncQ=truncQ,
                      maxLen=maxLen,
                      minLen=minLen,
                      maxN=maxN,
                      minQ=minQ,
                      rm.phix=rm.phix,
                      truncLen=truncLen,
                      trimLeft=trimLeft,
                      trimRight=trimRight,
                      maxEE=maxEE,
                      rm.lowcomplex=rm.lowcomplex,
                      compress=FALSE, verbose=TRUE)
  message(" ")
  message("The Filtered FASTQ Files have been saved in ", getwd()," as", Forward_F
iltered_File_Name, " and ",Reverse_Filtered_File_Name)
}}
}
```

F. ***FetchFlanks()***

This function is used to retrieve the sequences surrounding the variants for primer construction.

vcf_list	R object into which vcf data was stored with ReadVcf()
Reference_Genome	The reference genome unto which the sequence data has been mapped (including .fasta or .fna file extension).
validation	Whether or not the submitted reference genome is compared to data from the vcf_list to verify that the correct reference has been submitted. This prevents the function from potentially generating flanking regions for non-existing alleles.
Flank_Length	How many base pairs left and right of the variant are to be extracted. The default is 75 bp which produces a sequence of 150 bp (75 left of the variant and 75 right of the variant).
Output_Base_Name	To save the flanking regions directly to an excel file, insert the desired base file name here. The file extension is automatically added to this name.

Source Code

```
FetchFlanks<-function(vcf_list="No Default",
                      Reference_Genome="No Default",
                      validation=TRUE,
                      Flank_Length=75,
                      Output_Base_Name="No Default"){
# Check input
if(file.exists(Reference_Genome)==FALSE){return(message("No existing Reference gen
ome has been submitted"))
}

#Extract the sequence data
  message("Reading reference...")
seq<-seqinr::read.fasta(Reference_Genome)

if(validation==T){
# As a control, the ref alleles are called from the chromosome to see if these are
the same as the ones stated in the vcf file. This should reveal whether the used
fasta file is correct
    message("Verifying reference genome input")
  # Walk through all chromosomes
  i<-1
  while(i<=length(levels(as.factor(vcf_list[[1]]$CHROM)))){
    ref<-seq[[which(names(seq)==levels(as.factor(vcf_list[[1]]$CHROM))[i])]][vcf_l
ist[[1]]$POS[which(vcf_list[[1]]$CHROM==levels(as.factor(vcf_list[[1]]$CHROM))[i])
]]
    i<-i+1}

if(length(ref[is.element(tolower(ref),tolower(vcf_list[[1]]$REF))])==length(vcf_li
st[[1]]$REF)){
   message("Reference verified: The reference genome corresponds to the vcf data."
)
} else{return(message("Information of the reference and vcf file does not seem to
correspond. Please check that the correct file input was given."))}
}

# Calculate the start and ends of the flanking regions.
L_start<-vcf_list[[1]]$POS-Flank_Length
L_end<-vcf_list[[1]]$POS-1
R_start<-vcf_list[[1]]$POS+1
R_end<-vcf_list[[1]]$POS+Flank_Length

# Merge the limits for parsing
L<-paste(L_start,L_end,sep=":")
R<-paste(R_start,R_end,sep=":")

# Create the command
z<-1:length(vcf_list$General_Parameters$POS)
commandL<-paste("Left_Flank[[",z,"]]<-toupper(paste(seq[[which(names(seq)=='",vcf_
list[[1]]$CHROM,"')]][",L,"], collapse=''))",sep="")
commandR<-paste("Right_Flank[[",z,"]]<-toupper(paste(seq[[which(names(seq)=='",vcf
_list[[1]]$CHROM,"')]][",R,"], collapse=''))",sep="")

# Create empty vectors that will be filled with the flanks
Left_Flank<-list()
Right_Flank<-list()

# Parse the flanking calls
eval(parse(text = commandL))
eval(parse(text = commandR))
```

```
# Combine left and right flanks to receive the entire region
Sequence<-paste(unlist(Left_Flank),"[",vcf_list[[1]]$REF,"/",vcf_list[[1]]$ALT,"]"
,unlist(Right_Flank),sep="")

# Add Variant Name
Region_Name<-paste(vcf_list[[1]]$CHROM,vcf_list[[1]]$POS,sep="_")
Flanks<-cbind(Region_Name,Sequence)


# Optional: Save to excel
if(Output_Base_Name!="No Default"){
Output_Base_Name<-paste(Output_Base_Name,".xlsx",sep="")
openxlsx::write.xlsx2(Flanks,file=Output_Base_Name,sheetName="Flanking Regions",
                      col.names=T,row.names=F,append=F)}

# End of function
return(Flanks)
}
```

G. *GATKCall()*

This function allows the user to send commands to the GATK, running either natively on linux and Mac or within a docker for Windows, from within R.

Reference_Genome	The reference genome unto which the sequence data has been mapped (including .fasta or .fna file extension).
Alignment_File_Name	The file name of the alignment data from which variants are to be called (including .bam)
Output_Base_Name	The desired output base name for the variant call format (VCF) file. [default="Gatk_Call"]
gvcf.mode	Whether to store the variant call data in a genomic file (g.vcf). Use this when calling variants from multiple sample alignments that are to be merged later into one large vcf. [default=FALSE]
BAM_Validation	Whether to check for anomalies in the alignment file before performing variant calling. [default=FALSE]
BAM_Repair	Whether to repair or replace the alignment file header. Enable this if the .BAM file is malformed or if the sample name is incorrect by inserting the new header as a character. An example of such a header is "-SM Example -LB 1 -PL Illumina -PU 10578252" [default=FALSE]
pathtoGATK	Only for running GATK on linux: the file path to the GATK software when a custom installation path has been used. The default is ./gatk, which assumes gatk is added to your $PATH.
pathtoSamtools	Only for running GATK on linux when BamValidation=T or BamRepair=T: the file path to the samtools software when a custom installation path has been used. The default is ./samtools, which assumes samtools is added to your $PATH.
pathtobash	Only for running GATK within a docker (Mainly for windows users): the file path to git bash (bash.exe) when installed in a custom file location. Linux users can alter this directory path to make use of GATK within the docker environment as well.

Source Code

```
GATKCall<-function(Reference_Genome="No Default",
                   Alignment_File_Name="No Default",
                   Output_Base_Name="Gatk_Call",
                   gvcf.mode=FALSE, BAM_Validation=FALSE, BAM_Repair=FALSE,
                   pathtobash="C:/Program Files/Git/bin/bash.exe",
                   pathtoGATK="./gatk",
                   pathtoSamtools="./samtools"){
# Check Input
if(file.exists(Reference_Genome)==FALSE){return(message("No existing Reference gen
ome has been submitted"))
}else if(file.exists(Alignment_File_Name)==FALSE){return(message("No existing alig
nment file has been submitted"))
}else if(BAM_Repair==TRUE){return(message("BAM file repair has been turned on, but
no replacement header has been supplied. Please do so by inserting the new header
as a character behind BAM_Repair= "))
}else if(.Platform$OS.type=="windows" & file.exists("C:/Program Files/Git/bin/bash
.exe")==FALSE){return(message("Git bash could not be found at the defined location
. Change pathtobash to the correct location or install git bash and the docker fro
m https://docs.docker.com/toolbox/toolbox_install_windows/ or https://docs.docker.
com/engine/install/#server"))}

# Variant Calling Windows version
if(.Platform$OS.type=="windows" | pathtobash!="C:/Program Files/Git/bin/bash.exe")
{
  message("Using docker environment")

# Adapt windows file path to Unix commands
if(.Platform$OS.type=="windows"){
  wd<-paste("//",gsub(":/","/",getwd()),sep="")
  }else{wd<-getwd()}

# Add the file paths in front of input names
Reference_GenomeP<-paste(wd,"/",Reference_Genome,sep="")
Alignment_File_NameP<-paste(wd,"/",Alignment_File_Name,sep="")
pathtobash<-paste("'",pathtobash,"'",sep="")

# Start docker
  message("Calling upon docker")
shell(cmd = "'docker-machine restart'", shell=pathtobash, intern=T, flag = "-c")

# Check software installation
  message("Check for GATK and Samtools software")
shell(cmd = "'docker pull broadinstitute/gatk:4.1.3.0
      docker pull biocontainers/samtools:v1.9-4-deb_cv1'", shell=pathtobash, inter
n=T, flag = "-c")

# Prepare the reference genome if this has not yet been done
if(file.exists(paste(gsub("\\..*","",Reference_Genome),".dict",sep=""))==FALSE |
   file.exists(paste(gsub("\\..*","",Reference_Genome),".fai",sep=""))==FALSE){
  # Index the fasta file
    message("No previous index found. Indexing reference fasta file")
  command<-paste("'docker run -v ",wd,"://c/Users/Public biocontainers/samtools:v1
.9-4-deb_cv1 samtools faidx ",Reference_GenomeP,"'",sep="")
  shell(cmd = command, shell=pathtobash, intern=T, flag = "-c")

 if(file.exists(paste(gsub("\\..*","",Reference_Genome),".fai",sep=""))){
   message("The reference index has been made successfully")
  }else{message("Something went wrong and no index has been made. Please view the
samtools output above for details")}
```

```
  # Create Fasta Dictionary
    message("Creating reference fasta dictrionary")
  command<-paste("'docker run -v ",wd,"://c/Users/Public broadinstitute/gatk:4.1.3
.0 ./gatk CreateSequenceDictionary -R ",Reference_GenomeP," -O //c/Users/Public/",
gsub("\\..*","",Reference_Genome),"'",sep="")
  shell(cmd = command, shell=pathtobash, intern=T, flag = "-c")

  if(file.exists(paste(gsub("\\..*","",Reference_Genome),".dict",sep=""))){
    message("The reference dictionary has been made successfully")
  }else{message("No dictionary has been made. Something went wrong, please view th
e GATK output above for details")}}

if(BAM_Validation==T){
  # Validate Bam File
    message("Validating Alignment File")
  command<-paste("'docker run -v ",wd,"://c/Users/Public broadinstitute/gatk:4.1.3
.0 ./gatk ValidateSamFile -I ",Alignment_File_NameP," -MODE SUMMARY'",sep="")
  shell(cmd = command, shell=pathtobash, intern=T, flag = "-c")}

if(BAM_Repair!=F){
  message("Repairing Bam File")
command<-paste("'docker run -v ",wd,"://c/Users/Public broadinstitute/gatk:4.1.3.0
./gatk AddOrReplaceReadGroups -I ",Alignment_File_NameP," -O //c/Users/Public/",Al
ignment_File_Name," ",BAM_Repair,"'",sep="")
shell(cmd = command, shell=pathtobash, intern=T, flag = "-c")}

# Call Variants
if(gvcf.mode==FALSE){
  message("Building BAM index")
command<-paste("'docker run -v ",wd,"://c/Users/Public broadinstitute/gatk:4.1.3.0
./gatk BuildBamIndex -I ",Alignment_File_NameP,"'",sep="")
shell(cmd = command, shell=pathtobash, intern=T, flag = "-c")
  message(" ")
  message("Performing single sample variant calling")
Output_Base_Name<-paste(Output_Base_Name,".vcf",sep="")
command<-paste("'docker run -v ",wd,"://c/Users/Public broadinstitute/gatk:4.1.3.0
./gatk HaplotypeCaller -R ",Reference_GenomeP," -I ",Alignment_File_NameP," -O //c
/Users/Public/",Output_Base_Name,"'",sep="")
shell(cmd = command, shell=pathtobash, intern=T, flag = "-c")
  message("The variant call has been saved in ",getwd()," as ",Output_Base_Name)
}else{
  message("Building BAM index")
command<-paste("'docker run -v ",wd,"://c/Users/Public broadinstitute/gatk:4.1.3.0
./gatk BuildBamIndex -I ",Alignment_File_NameP,"'",sep="")
shell(cmd = command, shell=pathtobash, intern=T, flag = "-c")
  message(" ")
  message("Performing gVCF variant calling")
Output_Base_Name<-paste(Output_Base_Name,".g.vcf",sep="")
command<-paste("'docker run -v ",wd,"://c/Users/Public broadinstitute/gatk:4.1.3.0
./gatk HaplotypeCaller -R ",Reference_GenomeP," -I ",Alignment_File_NameP," -O //c
/Users/Public/",Output_Base_Name," -ERC GVCF'",sep="")
shell(cmd = command, shell=pathtobash, intern=T, flag = "-c")
  message("The variant call has been saved in ",getwd()," as ",Output_Base_Name)
}

# Shut down docker
  message("Variant calling has been performed. Shutting down...")
shell(cmd = "'docker-machine stop'", shell=pathtobash, intern=T, flag = "-c")
```

```
}else{
# Variant Calling Unix and Mac version
  message("Using native software")
# Add the file paths in front of input names
wd<-paste("//",gsub(":/","/",getwd()),sep="")
Reference_GenomeP<-paste(wd,"/",Reference_Genome,sep="")
Alignment_File_NameP<-paste(wd,"/",Alignment_File_Name,sep="")

# Prepare the reference genome if this has not yet been done
if(file.exists(paste(gsub("\\..*","",Reference_Genome),".dict",sep=""))==FALSE |
   file.exists(paste(gsub("\\..*","",Reference_Genome),".fai",sep=""))==FALSE){
# Index the fasta file
  message("No previous index found. Indexing reference fasta file")
command<-paste(pathtoSamtools," faidx ",Reference_GenomeP,sep="")
system(command, intern=TRUE)

# Create Fasta Dictionary
  message("Creating reference fasta dictrionary")
command<-paste(pathtoGATK," CreateSequenceDictionary -R ",Reference_GenomeP," ", g
etwd(), "/",gsub("\\..*","",Reference_Genome),sep="")
system(command, intern=TRUE)

if(file.exists(paste(gsub("\\..*","",Reference_Genome),".fai",sep=""))){
  message("The reference index has been made successfully")
}else{message("No index has been made. Something went wrong, please view the samto
ols output above for details")
  }
}

if(BAM_Validation==T){
  # Validate Bam File
  message("Validating Alignment File")
command<-paste(pathtoGATK," ValidateSamFile -I ",Alignment_File_NameP," -MODE SUMM
ARY",sep="")
system(command, intern=TRUE)

if(file.exists(paste(gsub("\\..*","",Reference_Genome),".dict",sep=""))){
    message("The reference dictionary has been made successfully")
}else{message("No dictionary has been made. Something went wrong, please view the
GATK output above for details")}
}

if(BAM_Repair!=FALSE){
      message("Repairing Bam File")
  command<-paste(pathtoGATK," AddOrReplaceReadGroups -I ",Alignment_File_NameP," "
, getwd(), "/",Alignment_File_Name," ",BAM_Repair,sep="")
  system(command, intern=TRUE)
}

# Call Variants
if(gvcf.mode==FALSE){
  message("Buidling BAM Index")
command<-paste(pathtoGATK," BuildBamIndex -I ",Alignment_File_NameP,sep="")
system(command, intern=TRUE)
  message(" ")
  message("Performing single sample variant calling")
Output_Base_Name<-paste(Output_Base_Name,".vcf",sep="")
command<-paste(pathtoGATK," HaplotypeCaller -R ",Reference_GenomeP," -I ",Alignmen
```

```
t_File_NameP," ", getwd(), "/",Output_Base_Name,sep="")
system(command, intern=TRUE)
  message("The variant call has been saved in ",getwd()," as ",Output_Base_Name)
}else{
  message("Buidling BAM Index")
command<-paste(pathtoGATK," BuildBamIndex -I ",Alignment_File_NameP,sep="")
system(command, intern=TRUE)
  message(" ")
  message("Performing gVCF variant calling")
Output_Base_Name<-paste(Output_Base_Name,".g.vcf",sep="")
command<-paste(pathtoGATK," HaplotypeCaller -R ",Reference_GenomeP," -I ",Alignmen
t_File_NameP," ", getwd(), "/",Output_Base_Name," -ERC GVCF'",sep="")
system(command, intern=TRUE)
    message("The variant call has been saved in ",getwd()," as ",Output_Base_Name)
}}}
```

H. *MergeAndGenotype()*

This function allows the user to send commands to the GATK, running either natively on linux and Mac or within a docker for Windows, from within R. This call merges g.vcf files into a final .vcf file.

Reference_Genome	The reference genome unto which the sequence data has been mapped (including .fasta or .fna file extension).
GVCF_Files	The g.vcf file names to merge into one. Input the names into a character string like so: c('one.g.vcf','two.g.vcf') or in a single string like so: "one.g.vcf two.g.vcf"
Output_Base_Name	The desired output base name for the variant call format (VCF) file. [default="Merged_GVCF"]
gvcf.mode	Whether to store the variant call data in a genomic file (g.vcf). Use this when calling variants from multiple sample alignments that are to be merged later into one large vcf. [default=FALSE]
pathtoGATK	Only for running GATK on linux: the file path to the GATK software when a custom installation path has been used. The default is ./gatk, which assumes gatk is added to your $PATH.
pathtobash	Only for running GATK within a docker (Mainly for windows users): the file path to git bash (bash.exe) when installed in a custom file location. Linux users can alter this directory path to make use of GATK within the docker environment as well.

Source Code

```
MergeAndGenotype<-function(Reference_Genome="No Default",
                    GVCF_Files="No Default",
                    Output_Base_Name="Merged_GVCF",
                    pathtoGATK='./gatk',
                    pathtobash="C:/Program Files/Git/bin/bash.exe"){
# Check Input
if(file.exists(Reference_Genome)==FALSE){return(message("No existing Reference gen
ome has been submitted"))
}else if(length(GVCF_Files)<=1){return(message("Submit at least two g.vcf files to
merge."))}

# Index input
if(length(GVCF_Files)==1 & grepl(" ",GVCF_Files)[1]==T){
  GVCF_Files<-unlist(strsplit(GVCF_Files," "))}
```

```
if(length(GVCF_Files)==1 & grepl(" , ",GVCF_Files)[1]==T){
  GVCF_Files<-unlist(strsplit(GVCF_Files," , "))}

GVCF_Files<-paste(paste("--variant ",GVCF_Files,sep=""),collapse=" ")
Output_Base_Name1<-paste(Output_Base_Name,"_temp.g.vcf",sep = "")
Output_Base_Name2<-paste(Output_Base_Name,".vcf",sep = "")

## Merge on Windows
if(.Platform$OS.type=="windows" | pathtobash!="C:/Program Files/Git/bin/bash.exe")
{
  message("Using docker environment")

# Adapt windows file path to Unix commands
if(.Platform$OS.type=="windows"){
  wd<-paste("//",gsub(":/","/",getwd()),sep="")
}else{wd<-getwd()}

# Add the quotes around bash
pathtobash<-paste("'",pathtobash,"'",sep="")

# Start docker
message("Calling upon docker")
shell(cmd = "'docker-machine restart'", shell=pathtobash, intern=T, flag = "-c")

# Check software installation
message("Check for GATK and Samtools software")
shell(cmd = "'docker pull broadinstitute/gatk:4.1.3.0'", shell=pathtobash, intern=
T, flag = "-c")
# Combine g.vcfs
  message("Combining g.vcf files...")
  message("")
command<-paste("'docker run -v ",wd,"://c/Users/Public broadinstitute/gatk:4.1.3.0
./gatk CombineGVCFs -R ",Reference_Genome," ",GVCF_Files," -O ",Output_Base_Name1,
"'",sep="")
shell(cmd = command, shell=pathtobash, intern=T, flag = "-c")

# Genotype combined file
  message("Genotyping results and converting to vcf...")
  message("")
command<-paste(pathtoGATK," GenotypeGVCFs -R ",Reference_Genome," -V ",Output_Base
_Name1," -O ",Output_Base_Name2)
shell(cmd = command, shell=pathtobash, intern=T, flag = "-c")
  message("Merging has been completed. The final file as been saved as ",Output_Ba
se_Name2," in ",getwd())

# Shut down docker
  message("Shutting down...")
shell(cmd = "'docker-machine stop'", shell=pathtobash, intern=T, flag = "-c")

}else{
## Merge on linux and Mac
# Combine g.vcfs
  message("Combining g.vcf files...")
  message("")
command<-paste(pathtoGATK," CombineGVCFs -R ",Reference_Genome," ",GVCF_Files," -O
",Output_Base_Name1,sep="")
system(command, intern=TRUE)
# Genotype combined file
  message("Genotyping results and converting to vcf...")
```

```
  message("")
command<-paste(pathtoGATK," GenotypeGVCFs -R ",Reference_Genome," -V ",Output_Base
_Name1," -O ",Output_Base_Name2)
  message("Merging has been completed. The final file as been saved as ",Output_Ba
se_Name2," in ",getwd())
}}
```

l. *ReadVcf()*

Based on vcfR, this function is used to read variant data from a variant call format (VCF) file into a comprehensive list of dataframes. This list is split into general parameters and sample specific parameters.

vcf_file	The file to be read in (including .vcf file extension)
contig	The name of the contig that has to be read in. For example "C1" or "NC_044370.1". Note that nothing is read in if the contig name is misspelled and only one contig name can be submitted.
nrows	The number of rows. The Scan_Vcf function returns the nrows per chromosome.
skip	The number of rows to skip before starting to read. The Scan_Vcf function returns the skip number corresponding with the chromosome of interest.
sample_nr	The number(s) of the sample(s) to be included. The first sample always has sample_nr 10 and the default sample_nr 0 reads in all samples present. Specific sample numbers are returned with the Scan_Vcf function

Source Code

```
ReadVcf<-function(vcf_file="No Default",
                  contig="No Default",
                  nrows=-1,
                  skip=0,
                  sample_nr=0,
                  verbose=TRUE){
if(file.exists(vcf_file)==F){return(message("No input vcf file has been submitted
or the submitted file does not exist in the working directory."))
}

## Read in vcfR object
# Collect Stats
stats<-vcfR::.vcf_stats_gz(vcf_file,nrows=nrows,skip=skip,verbose=verbose)
# Inventorise how many columns have to be read in
if(sample_nr==0){
  cols<-sort(unique((1:stats[4])))
}else{
  cols<-sort(unique(c(1:9,sample_nr)))}
# Read the data
  message("Reading in data...")
  body<-vcfR::.read_body_gz(vcf_file,stats=stats,nrows=nrows,skip=skip,cols=cols,c
onvertNA=as.numeric(1),verbose=as.numeric(verbose))
# Select relevant contigs
if(contig!="No Default"){
  c<-grep(contig,body[,1])
  c<-c[length(c)]
  body<-body[c[1]:c[length(c)],]
}
```

```
## Order the vcf data into a matrix
# Labelling variant type
a<-cbind(which(nchar(body[,4])==1),"SNP")
b<-cbind(which(nchar(body[,4])>1),"MNP")
if(ncol(a)==1){
  # No SNP
  c<-as.data.frame(b)
}else if(ncol(b)==1){
  c<-as.data.frame(a)
  # No MNP
} else{
c<-as.data.frame(rbind(a,b));c[,1]<-as.numeric(as.character(c[,1]));c[,2]<-as.char
acter(c[,2])}

Type<-c[order(c[,1]),2]
# Info column sorting
  message("Sorting Info columns")
info_sort<-list()
if(length(which(grepl("END=",body[,8])!=T))>0){
  # Info column is not empty
  tX<-as.data.frame(body[,8]);tX[,1]<-as.character(tX[,1])
  ref<-body[which((stringr::str_count(body[,8], ";") + 1) == max(stringr::str_coun
t(body[,8], ";") + 1))[1],8]
  ref<-unlist(strsplit(ref,";"))
  ref<-gsub("=.*","",ref)

  if(length(body[which((stringr::str_count(body[,8], ";") + 1) != max(stringr::str
_count(body[,8], ";") + 1)),8])==0){
    # No missing parameters
    tX[,1]<-gsub(";",":",tX[,1])
  }else{
    # Some parameters are missing
    if(length(tX[which(stringr::str_count(body[,8], ref[1])<1),1])>0){
      # Replace first missing parameter
      tX[which(stringr::str_count(body[,8], ref[1])<1),1]<-sub("","NA;",tX[which(s
tringr::str_count(body[,8], ref[1])<1),1])}
    z<-2
    while(z<=length(ref)){
      # Replace non-first missing parameters
      tX[which(stringr::str_count(body[,8], ref[z])<1),1]<-sub(";",":NA;",tX[which
(stringr::str_count(body[,8], ref[z])<1),1])
      # Log what parameter has been checked
      tX[which(stringr::str_count(body[,8], ref[z])>=1),1]<-sub(";",":",tX[which(s
tringr::str_count(body[,8], ref[z])>=1),1])
      z<-z+1
    }}
  # Separate
  tX<-tidyr::separate(tX, col=1, into=ref,sep=":")
  # General removal of parameter names
  z<-1
  while(z<=length(ref)){
    tX[,z]<-gsub(paste(ref[z],"=",sep=""),"",tX[,z])
    z<-z+1
  }
  tX[,length(ref)]<-gsub(";","",tX[,length(ref)])
  # Addition of new info column to the body
  info_sort[[1]]<-cbind(body[,1:3],Type,body[,4:7],tX)
  names(info_sort)[[1]]<-"General_Parameters"
}else{info_sort<-cbind(body[,1:3],Type,body[,4:8])}
```

```
# Sample Specific ordering:
# Prepare an output
  message("Sorting Sample Specific columns")
Sample_Specific_Parameters<-list()
if(ncol(body)>9){
  # Get full parameter list
  ref<-body[which((stringr::str_count(body[,9], ":") + 1) == max(stringr::str_coun
t(body[,9], ":") + 1))[1],9]
  ref<-unlist(strsplit(ref,":"))
  # Process each sample
  i<-1
  col<-10
  while(col<=ncol(body)){
    # Process sample specific data
    tX<-as.data.frame(body[,col]);tX$`body[, col]`<-as.character(tX$`body[, col]`)
    if(length(body[which((stringr::str_count(tX[,1], ":") + 1) != max(stringr::str
_count(body[,9], ":") + 1))[1],9])==0){
      # No parameters are missing
      tX<-tidyr::separate(tX, col=1, into=ref,sep=":")
    }else{
      # Some parameters are missing
      if(length(tX[which(stringr::str_count(body[,9], ref[1])<1),1])>0){
        # Replace first missing parameter
        tX[which(stringr::str_count(body[,9], ref[1])<1),1]<-sub("","NA:",tX[which
(stringr::str_count(body[,9], ref[1])<1),1])}
      z<-2
      while(z<=length(ref)){
        # Replace non-first missing parameters
        tX[which(stringr::str_count(body[,9], ref[z])<1),1]<-sub(":",";NA:",tX[whi
ch(stringr::str_count(body[,9], ref[z])<1),1])
        # Log what parameter has been checked
        tX[which(stringr::str_count(body[,9], ref[z])>=1),1]<-sub(":",";",tX[which
(stringr::str_count(body[,9], ref[z])>=1),1])
        z<-z+1
      }
      # Replace the remainder of the parameter separators and split into columns
      tX[,1]<-gsub("(;\\.)",";NA",tX[,1])
      suppressWarnings(tX<-tidyr::separate(tX, col=1, into=ref,sep=";"))
    }

    # Store work and prepare for the next sample
    Sample_Specific_Parameters[[i]]<-tX
    names(Sample_Specific_Parameters)[i]<-colnames(body)[col]
      message("Processed sample ",i," out of ",ncol(body)[1]-9)
    i<-i+1
    col<-col+1}

  # End of samples loop
}

# Compile
output<-c(info_sort,Sample_Specific_Parameters)
# Change parameters into R numerics and characters
  message("Changing into R characters and numericals")
z<-1
while(z<=length(output)){
  i<-1
  while(i<=ncol(output[[z]])){
```

```
    if(length(output[[z]][which(grepl("[B|C|D|F|G|H|I|J|K|L|M|O|P|Q|R|S|T|U|V|W|X|
Y|Z|:|,|//|//|]",unname(output[[z]][,i]),ignore.case = T)==T),i])>0){
      # The data contains non-numerics
      output[[z]][,i]<-as.character(output[[z]][,i])
      i<-i+1
    }else{
      # The data contains only numerics
      output[[z]][,i]<-suppressWarnings(as.numeric(as.character(output[[z]][,i])))
      i<-i+1
    }}
  z<-z+1}

# End of function:
return(output)}
```

J. *SaveVcfObject()*

This function allows the user to save a vcf R object created by readVcf() into an excel file where sample specific parameters are stored in separate sheets. Note that this saving method is only functional for small datasets and not whole genome data.

vcf_list	The R object into which the vcf data was stored by readVcf.
Output_Base_Name	The desired output base name for the created excel file. [default="VcfObject"]

Source Code

```
SaveVcfObject<-function(vcf_list="No Default",
                        Output_Base_Name="VcfObject"){
# Check input
if(file.exists(vcf_list)=="No Default"){return(message("No vcf R object has been s
ubmitted"))
}
Output_Base_Name<-paste(Output_Base_Name,".xlsx",sep="")
# General parameters
  message("Saving General Parameters")
xlsx::write.xlsx2(vcf_list[[1]], Output_Base_Name, sheetName = "General Parameters
",
              col.names = TRUE, row.names = FALSE, append = FALSE)
# Sample Specific parameters
  message("Saving Sample Specific Parameters")
i<-2
while(i<=length(vcf_list)){
xlsx::write.xlsx2(vcf_list[[i]], Output_Base_Name, sheetName = names(vcf_list)[i],
                  col.names = TRUE, row.names = FALSE, append = TRUE)
i<-i+1
}

# End of function
  message("The vcf R object has been saved as ",Output_Base_Name," in ",getwd())
}
```

K. Scan_Vcf()

Based on VariantAnnotation, this function allows the user to find the starting row and number of variants of all chromosomes in a vcf file. This function is meant to provide an overview of the number of variants called, as well as the nrows and skip arguments required when reading in a vcf per chromosome.

vcf_file	The Variant Call Format (VCF) file name to scan (including .vcf extension). This function is incompatible with .gz compressed file.

Source Code

```
#' Scan_Vcf()
#'
#' #' @param
#' @keywords exploration scan vcf
#' @export
#' @examples
#' # Scan a vcf file:
#' Scan_Vcf("file.vcf")

Scan_Vcf<-function(vcf_file="No Default"){
# Check input
if(file.exists(vcf_file)==F){return(message("No input vcf file has been submitted
or the submitted file does not exist in the working directory."))
}else if(grepl(".gz",vcf_file)==1){return(message("Please decompress the vcf file
before submission."))}

# Check input
if(file.exists(vcf_file)==F){return(message("No input vcf file has been submitted
or the submitted file does not exist in the working directory."))
}else if(grepl(".gz",vcf_file)==1){return(message("Please decompress the vcf file
before submission."))}

# Find chromosome names
hdr<-suppressWarnings(VariantAnnotation::scanVcfHeader(vcf_file))

# Store as dataframe
Chr_list<-data.frame(hdr@header@listData$contig@rownames)
names(Chr_list)<-"Contig Name"

# Find sample names and numbers
Samples<-as.data.frame(hdr@samples);colnames(Samples)<-("Sample_Name")
Sample_Nr<-as.numeric(rownames(Samples))+9
Samples<-cbind(Samples,Sample_Nr)
# Output the dataframe
output<-list(Chr_list,Samples)
names(output)<-c("Chromosomes","Samples")
return(output)}
```

L. ***VariantFilter()***

This function enables filtering of variant data in a R object based on various quality criteria

vcf_list	R object into which vcf data was stored with ReadVcf()
General_filter	param The filter criteria for general parameters, defined as a character string per criterium. View examples for details
Sample_Specific_Filter	The filter criteria for sample parameters, defined as a character string per criterium. View examples for details
min_dist	The minimum variant free space around any given variant in basepairs. Use this when the presence of nearby variants is undesirable, for example during variant genotyping primer construction. This filtering criterium takes all variants of the vcf into account. If only high quality variants should be taken into account, perform the filtering step twice.

min_AD	The minimum summed allelic depth allowed. This is a measure of coverage used to discard variant with little evidence.
max_AD	The maximum summed allelic depth allowed. This is a measure of oversequencing often observed for repeats and overabundant transposons.

Source Code

```
VariantFilter<-function(vcf_list="No Default",
                        General_filter="No Default",
                        Sample_Specific_Filter="No Default",
                        min_dist="No Default",
                        min_AD=0,
                        max_AD=Inf){

# Check input
if(exists("vcf_list")==F){return(message("No vcf data has been submitted"))
}else if(General_filter[1]=="No Default" & Sample_Specific_Filter[1]=="No Default"
&min_AD==0 &max_AD==Inf&min_dist=="No Default"){return(message("No filtering crite
ria were given"))}

# Setup output
i<-1
f<-list()
filt<-list()
filt1<-list()

# Filter based on general and sample specific parameters
if(General_filter[1]!="No Default"&Sample_Specific_Filter[1]!="No Default"){
# Index specific filters
z<-2
while(z<=length(vcf_list)){ #While for sample
    f<-paste("vcf_list[[",z,"]]$",Sample_Specific_Filter,sep="")
    f<-paste("filt1[[",z-1,"]]<-which(",paste(f,collapse = " & "),")",sep = "")
    # Look for passing values
    eval(parse(text = f))
z<-z+1}

# Index general filters
f<-paste("vcf_list[[1]]$",General_filter,sep="")
f<-paste("filt[[",z,"]]<-which(",paste(f,collapse = " & "),")",sep = "")
# Look for passing values
eval(parse(text = f))
# Index passed variant row numbers
filt<-append(unlist(filt1),unlist(filt))
filt<-filt[duplicated(filt)] #Keep only duplicates
filt<-filt[!duplicated(filt)] #Remove extra duplicates

# Filter based on sample specific parameters only
}else if(General_filter[1]=="No Default"&Sample_Specific_Filter[1]!="No Default"){
  # Index specific filters
  z<-2
  while(z<=length(vcf_list)){
    f<-paste("vcf_list[[",z,"]]$",Sample_Specific_Filter,sep="")
  f<-paste("filt[[",z-1,"]]<-which(",paste(f,collapse = " & "),")",sep = "")
  # Look for passing values
  eval(parse(text = f))
  z<-z+1}
  # Index passed variant row numbers
```

```
  filt<-unlist(filt)
# Filter based on general parameters only
}else if(General_filter[1]!="No Default"&Sample_Specific_Filter[1]=="No Default"){
  # Index general filters
  f<-paste("vcf_list[[1]]$",General_filter,sep="")
  f<-paste("filt<-which(",paste(f,collapse = " & "),")",sep = "")
  # Look for passing values
  eval(parse(text = f))
  # Index passed variant row numbers
  filt<-unlist(filt)
# End of filter
}else{filt<-unlist(filt)}

# Check for nearby variants (for primer development)
if(min_dist!="No Default"){
  # Calculate distance between variants
  Dist_Raw<-diff(vcf_list$General_Parameters$POS)
  Dist_Raw1<-c(Dist_Raw,1000)
  Dist_Raw2<-c(1000,Dist_Raw)
  # Remove variants that are too close to each other
  bp_filt<-which(Dist_Raw1>=min_dist &
                 Dist_Raw2>=min_dist)
  # Add the passing variants to the filtering list (filt)
  bp_filt<-unlist(bp_filt)
  bp_filt<-bp_filt[duplicated(bp_filt)]
  bp_filt<-bp_filt[!duplicated(bp_filt)]
  filt<-append(filt,bp_filt)
  filt<-filt[duplicated(filt)] #Keep only duplicates
  filt<-filt[!duplicated(filt)] #Remove extra duplicates
}

# Filter on allelic depth
if(min_AD!=0 | max_AD!=Inf){
  f<-list()
  z<-2
  while (z<=length(vcf_list)) {
    AD_Sum<-0
    tX<-paste("AD_Sum<-c(AD_Sum,",gsub(",","+",vcf_list[[z]]$AD),")",sep="")
    eval(parse(text=tX))
    AD_Sum<-AD_Sum[-1]
    f[[z-1]]<-which(AD_Sum>=min_AD & AD_Sum<=max_AD)
    z<-z+1
  }
  f<-unlist(f)
  f<-f[duplicated(f)]
  f<-f[!duplicated(f)]
  filt<-append(filt,f)
  filt<-filt[duplicated(filt)] #Keep only duplicates
  filt<-filt[!duplicated(filt)] #Remove extra duplicates
}

# Store only variants that passed all filters
i<-1
pass <- list()
while(i<=length(vcf_list)){
pass[[i]]<-vcf_list[[i]][filt,]
i<-i+1
names(pass)<-names(vcf_list)
}
```

```
  # End of function
  return(pass)}
```

M. *VariantPlot()*

Based on ggplot2, this function is used to plot relevant quality parameters of variants stored in an R object with ReadVcf()

vcf_list	R object into which vcf data was stored with ReadVcf()
plot	param If only a specific parameter has to be plotted, insert the name with plot='parameter name'. This only works for the parameters plotted when plot='No Default'. Other parameters give an empty plot.
save_name	The generated quality plot can be be saved as a pdf file by setting save to the desired base save name (without .pdf extension)
server	To save the quality plots on a server without gpu access, set this variable to TRUE to use an alternative save method.

Source Code

```
VariantPlot<-function(vcf_list="No Default",
                      plot="No Default",
                      save_name="No Default",
                      server=FALSE){

# Check input
if(exists("vcf_list")==F){return(message("No vcf data has been submitted"))
  }else if(class(vcf_list)!="list"){return(message("The given object does not exis
t or is not of type list constructed with readVcf()"))}


# Plot available data and store into objects to later arrange
i<-1
y<-list()
# General information on position
if(plot=="No Default"){
  y[[i]]<-(ggplot2::ggplot(data=vcf_list[[1]],ggplot2::aes(x=POS))
           +ggplot2::geom_density(stat="bin",ggplot2::aes(),fill="dodgerblue3")
           +ggplot2::labs(title="Variant Distribution", y="Number of Variants",x="
Position (bp)")
           +ggplot2::coord_cartesian(expand = FALSE))
  i<-i+1
  y[[i]]<-(ggplot2::ggplot()
           +ggplot2::geom_density(alpha=0.5,stat="bin",ggplot2::aes(x=vcf_list[[1]
]$POS[which(vcf_list[[1]]$Type=="SNP")],fill="SNP"))
           +ggplot2::geom_density(alpha=0.5,stat="bin",ggplot2::aes(x=vcf_list[[1]
]$POS[which(vcf_list[[1]]$Type=="MNP")],fill="MNP"))
           +ggplot2::scale_fill_manual(labels=c("INDELs","SNPs"),values=c("SNP"="d
odgerblue2","MNP"="blue3"))
           +ggplot2::labs(title="Variant Distribution Per Type", y="Number of Vari
ants",x="Position (bp)",fill="Type")
           +ggplot2::theme(legend.title = ggplot2::element_blank(),legend.position
=c(0.9,0.86),legend.background=ggplot2::element_rect(fill=NA))
           +ggplot2::coord_cartesian(expand = FALSE))
  i<-i+1
  y[[i]]<-(ggplot2::ggplot()
           +ggplot2::geom_bar(stat="count",ggplot2::aes(x=as.factor(vcf_list[[1]]$
Type)),fill=c("dodgerblue2","blue3"))
           +ggplot2::geom_text(stat="count",data=vcf_list[[1]], ggplot2::aes(x=as.
```

```
factor(Type),colour="text",label=..count..), vjust=1.5)
           +ggplot2::scale_colour_manual(values = "white")
           +ggplot2::labs(title="Variant Type Numbers", y="Number of Variants",x="
Type")
           +ggplot2::theme(legend.position = "none")
           )
}

# Non standard information
if((length(grep("MQ",colnames(vcf_list[[1]])))!=0&plot=="No Default") | plot=="MQ"
){
  # Mapping Quality of the variant
  i<-i+1
  y[[i]]<-(ggplot2::ggplot(data=vcf_list[[1]],ggplot2::aes(x=MQ))
           +ggplot2::geom_density(stat="bin",ggplot2::aes(),fill="darkorchid4")
           +ggplot2::labs(title="Mapping Quality", y="Number of Variants",x="MQ sc
ore")
           +ggplot2::coord_cartesian(expand = FALSE))}
if((grep("MQRankSum",colnames(vcf_list[[1]])))!=0&plot=="No Default") | plot=="MQRa
nkSum"){
  # Mapping Quality Ranksum
  i<-i+1
  y[[i]]<-(ggplot2::ggplot(data=vcf_list[[1]],ggplot2::aes(x=MQRankSum))
           +ggplot2::geom_density(stat="bin",ggplot2::aes(),fill="darkorchid3")
           +ggplot2::labs(title="Mapping Quality of REF vs ALT Allele Reads", y="N
umber of Variants",x="MQRanksum")
           +ggplot2::coord_cartesian(expand = FALSE))}
if((grep("QD",colnames(vcf_list[[1]])))!=0&plot=="No Default") | plot=="QD"){
  # Quality normalised to depth
  i<-i+1
  y[[i]]<-(ggplot2::ggplot(data=vcf_list[[1]],ggplot2::aes(x=QD))
           +ggplot2::geom_density(stat="bin",ggplot2::aes(),fill="darkorchid2")
           +ggplot2::labs(title="QD distribution", y="Number of Variants",x="QD sc
ore")
           +ggplot2::coord_cartesian(expand = FALSE))}
if((grep("FS",colnames(vcf_list[[1]])))!=0&plot=="No Default") | plot=="FS"){
  # Fisher Strand Bias
  i<-i+1
  y[[i]]<-(ggplot2::ggplot(data=vcf_list[[1]],ggplot2::aes(x=FS))
           +ggplot2::geom_density(stat="bin",ggplot2::aes(),fill="deeppink3")
           +ggplot2::labs(title="Fisher Strand Bias", y="Number of Variants",x="FS
score")
           +ggplot2::coord_cartesian(expand = FALSE))}
if((grep("ReadPosRankSum",colnames(vcf_list[[1]])))!=0&plot=="No Default") | plot==
"ReadPosRankSum"){
  # Read Position Ranksum
  i<-i+1
  y[[i]]<-(ggplot2::ggplot(data=vcf_list[[1]],ggplot2::aes(x=ReadPosRankSum))
           +ggplot2::geom_density(stat="bin",ggplot2::aes(),fill="deeppink2")
           +ggplot2::labs(title="Allele Position Within Reads", y="Number of Varia
nts",x="ReadPosRankSum")
           +ggplot2::coord_cartesian(expand = FALSE))}
if((grep("SOR",colnames(vcf_list[[1]])))!=0&plot=="No Default") | plot=="SOR"){
  # Strand Odds Ratio
  i<-i+1
  y[[i]]<-(ggplot2::ggplot(data=vcf_list[[1]],ggplot2::aes(x=ReadPosRankSum))
           +ggplot2::geom_density(stat="bin",ggplot2::aes(),fill="deeppink1")
           +ggplot2::labs(title="Strand Odds Ratio", y="Number of Variants",x="Str
and Odds Ratio score")
```

```
          +ggplot2::coord_cartesian(expand = FALSE))}

# Arrange plots:
z<-suppressMessages(ggpubr::ggarrange(plotlist = y))
# Save plots if desired
if(save_name!="No Default"&server==F){
  save_name<-paste(save_name,".pdf",sep="")
  pdf(file=file.path(getwd(),save_name), width=11.7, height=11.7)
  z
  dev.off()
}else if(save_name!="No Default"&server==T){
 save_name<-paste(save_name,".pdf",sep="")
 bitmap(file=file.path(getwd(),save_name), width=11.7, height=11.7, type="pdfwrite
", res=100)
 z
 dev.off()
}
if(server==F){
# Print plots
suppressMessages(z)}

# End of function
}
```

N. *VariantToolsCall()*

This function allows the user to send commands to the GATK, running either natively on linux and Mac or within a docker for Windows, from within R.

Reference_Genome	The reference genome unto which the sequence data has been mapped (including .fasta or .fna file extension).
Alignment_File_Name	The file name of the alignment data from which variants are to be called (including .bam)
Output_Base_Name	The desired output base name for the variant call format (VCF) file. [default="Gatk_Call"]
minBaseQuality	Minimum nucleotide quality below which a variant will be masked. [default=10]
minMapQuality	Minimum mapping quality below which a variant will be masked. [default=10]
minDepth	Minimum read depth below which a variant will be masked [default=2]
maxDepth	Maximum read depth above which a variant will be masked [default=0.2]
p_lower	From VariantTools: 'The lower bound on the binomial probability for a true variant.'
p_error	From VariantTools: 'The binomial probability for a sequencing error (default is reasonable for Illumina data with the default quality cutoff).' [default=0.001]
read_count	From VariantTools: 'Require at least this many high quality reads with the alternate base. The default value is designed to catch sequencing errors where coverage is too low to rely on the LRT. Increasing this value has a significant negative impact on power.' [default=2]

Source Code

```r
VariantToolsCall<-function(Reference_Genome="No Default",
                           Alignment_File_Name="No Default",
                           Output_Base_Name="VariantToolsCall.vcf",
                           minBaseQuality=10,
                           minMapQuality=10,
                           minDepth=2,
                           maxDepth=Inf,
                           p_lower=0.2,
                           p_error=0.001,
                           read_count=2){
# Check input
if(file.exists(Reference_Genome)==FALSE){return(message("No existing Reference gen
ome has been submitted"))
}else if(file.exists(Alignment_File_Name)==FALSE){return(message("No existing alig
nment file has been submitted"))
}
  message("Locating Reference Bounderies")
# Fetch Reference
Ref<-Biostrings::readDNAStringSet(Reference_Genome)
Ref<-Biostrings::getSeq(Ref,names=Ref[1]@ranges@NAMES)

# Set Genomic Range to the entire genome
Genomic_Range<-GenomicRanges::GRanges(seqnames=Ref[1]@ranges@NAMES,
                                      ranges=IRanges(Ref[1]@ranges@start,Ref[1]@ra
nges@width))

# Register filter criteria
  message("Establishing Filtering Criteria")
param<-Rsamtools::ApplyPileupsParam(minBaseQuality=minBaseQuality,
                                    minMapQuality=minMapQuality,
                                    minDepth=minDepth,
                                    maxDepth=maxDepth,
                                    which=Genomic_Range,
                                    yieldAll=FALSE)

  message("Locating relevant information")
# Fetch BAM File
BAM<-Rsamtools::BamFile(Alignment_File_Name)
BAM<-Rsamtools::PileupFiles(Alignment_File_Name,param=param)

# Create a pileup file of those positions that passed the filters
  message("Filtering BAM File. This may take a while")
Pileup<-VariantTools::pileupVariants(BAM,Ref, param=param, baseOnly=TRUE)

# Filter
Calling_Filters<-VariantTools::VariantCallingFilters(p.lower=p_lower,
                                                     p.error=p_error,
                                                     read.count=read_count)
# Call variants
  message("Calling variants")
Called_Variants<-VariantTools::callVariants(Pileup,
                                            calling.filters=Calling_Filters)

# Save to a vcf file
  message("Saving to VCF File format")
vcfR::writeVcf(Called_Variants,Output_Base_Name)
  message("The variants have been saved as ",Output_Base_Name ," in ",getwd())
}
```

4. Example data generation

Due to time constraints, the randomly generated example data could not be constructed according to real-life paired end sequence data mappable to a reference. The underlying cause for being unable to align these data could not be discovered and as such, the `GenerateExample()` function was not implemented into the package. However, the example data are functional for testing quality plotting and raw data trimming. Therefore, the function code is given here in the appendix to support future users and coders that wish to use randomly generated data as a testing basis for the first two steps of variant calling.

object with ReadVcf()

Number_of_reads	The amount of reads sampled from the entire example genome, stored in a fastq file. [default=1000]
Number_of_variants	param The amount of variants hidden in the example data. This will be rounded to a number dividable by 4 in order to equally distribute the variants per nucleotide type. Half of the variants will be heterozygous. [default=40]
Length_of_genome	The basepair length of the example genome. Choosing a large number may cause the function to become extremely slow. [default=1000]
Read_length	The basepair length of a single read. The default corresponds newer with Illumina read lengths [default=150]
Paired_End_Data	param Whether to generate single or paired end data, where FALSE corresponds with single-ends [default=TRUE]

Source Code

```
GenerateExample<-function(Number_of_reads=1000,
                          Number_of_variants=40,
                          Length_of_genome=1000,
                          Read_length=150,
                          Paired_End_Data=TRUE){
# Construct S4 Class object to save output to
setClass(Class="Example_Data",
         representation(reference="character",
                        alternative="character"))
# Make read length even
if(grepl("\\.",Read_length/2)==T){Read_length<-Read_length+1}
# Construct a Reference
Reference_Genome<-paste(sample(Biostrings::DNA_ALPHABET[1:4], size=Length_of_genom
e, replace=TRUE),collapse="")
# Save Reference
seqinr::write.fasta(Reference_Genome,"example_reference",file.out = "example.fasta
")
  message("The example reference genome has been saved as example.fasta in ",getwd
())
  message("")
# Construct and Alternative Genome
Alt_Genome<-unlist(strsplit(Reference_Genome,""))
# Select random postions for the variants and convert half into heterozygous varia
nts
Number_of_variants<-round(Number_of_variants/4)*4
```

```
Variant_position<-c(sample(which(Alt_Genome!="A")[which(which(Alt_Genome!="A")
                    >=Read_length+1 &
                    which(Alt_Genome!="A")<=Length_of_genome-Read_length+1)],
                           Number_of_variants/4),
                    sample(which(Alt_Genome!="T")[which(which(Alt_Genome!="T")
                    >=Read_length+1 &
                    which(Alt_Genome!="T")<=Length_of_genome-Read_length+1)],
                           Number_of_variants/4),
                    sample(which(Alt_Genome!="G")[which(which(Alt_Genome!="G")
                           >=Read_length+1 &
                    which(Alt_Genome!="G")<=Length_of_genome-Read_length+1)],
                           Number_of_variants/4),
                    sample(which(Alt_Genome!="C")[which(which(Alt_Genome!="C")
                    >=Read_length+1 &
                    which(Alt_Genome!="C")<=Length_of_genome-Read_length+1)],
                           Number_of_variants/4))
Heterozygous_variants<-sample(Variant_position,Number_of_variants/2)
# Insert the variants into the chosen basepair positions
  message("Inserting variants...")
  message("")
Alt_Genome[Variant_position[1:(Number_of_variants/4)]]<-"A"
Alt_Genome[Variant_position[((Number_of_variants/4)+1):((Number_of_variants/4)*2)]
]<-"T"
Alt_Genome[Variant_position[((Number_of_variants/4)*2+1):((Number_of_variants/4)*3
)]]<-"G"
Alt_Genome[Variant_position[((Number_of_variants/4)*3+1):Number_of_variants]]<-"C"
# Store the variant positions in Example_Variants.txt
ALT<-c(replicate(Number_of_variants/4, "A"),replicate(Number_of_variants/4, "T"),
       replicate(Number_of_variants/4, "G"),replicate(Number_of_variants/4, "C"))
Type<-replicate(Number_of_variants, "homozygous")

Example_Variants<-cbind(Variant_position, ALT, Type)
Example_Variants[which(is.element(Example_Variants[,1],Heterozygous_variants)),3]<
-"heterozygous"
colnames(Example_Variants)<-c("Pos","ALT","Type")

write.table(Example_Variants,"Example_Variants.txt",sep="\t",row.names=FALSE)
  message("The variant postions and types have been saved in Example_Variants.txt"
)
  message("")
# Collapse the alternative genome
Alt_Genome<-paste(Alt_Genome,collapse="")
# Generate read qualities between 10 and 40
fw_quality<-replicate(Number_of_reads,{paste(sample(unlist(strsplit(rawToChar(as.r
aw(64:73)),"")),Read_length,replace=TRUE), collapse="")})
rv_quality<-replicate(Number_of_reads,{paste(sample(unlist(strsplit(rawToChar(as.r
aw(64:73)),"")),Read_length,replace=TRUE), collapse="")})
# Sample forward sequence reads
  message("Creating forward reads...")
  message("")
start<-sample(1:(Length_of_genome-Read_length),(length(fw_quality)-Number_of_varia
nts),replace=TRUE)
end<-start+Read_length-1 ; fw_reads<-list() ; i<-1 ; z<-1

repeat{
  if(i<=(length(fw_quality)-Number_of_variants)){
    fw_reads[i]<-substr(Alt_Genome,start=start[i],stop=end[i])
    i<-i+1
  } else if(i>=(length(fw_quality)-Number_of_variants) & i<=Number_of_reads){
```

```
    fw_reads[i]<-substr(Reference_Genome,start=Heterozygous_variants[z]-Read_lengt
h/2, stop=Heterozygous_variants[z]+(Read_length/2)-1)
    fw_reads[i+1]<-substr(Reference_Genome,start=Heterozygous_variants[z]-Read_len
gth/2, stop=Heterozygous_variants[z]+(Read_length/2)-1)
    i<-i+2;z<-z+1
  } else{rm(i,z,start,end)
    fw_reads<-unlist(fw_reads)
    break}
  }
# write a fastq record
  message("Saving Reads...")
names<-paste("example", 1:Number_of_reads)
suppressWarnings(ChIPsim::writeFASTQ(fw_reads, fw_quality,name=names, file="exampl
e_fw.fastq"))
  message("Forward reads have been saved in example_fw.fastq")

if(Paired_End_Data==TRUE){
    message("")
    message("Generating reverse reads...")
    message("")
  i<-1
  rv_reads<-list()
  # Reverse Reads
  while (i<=Number_of_reads) {
    rv_reads[[i]]<-paste(unlist(rev(strsplit(fw_reads[i],"")[[1]])),collapse="")
    i<-i+1}
  # write a fastq record
    message("Saving Reads...")
  suppressWarnings(ChIPsim::writeFASTQ(rv_reads, rv_quality,"example", file="examp
le_rv.fastq"))
    message("Reverse reads have been saved in example_rv.fastq")

  # Load example files
  Alternative_Genome<-c("example_fw.fastq","example_rv.fastq")
  Reference_Genome<-"example.fasta"
}else{
  # Load example files
  Alternative_Genome<-c("example_fw.fastq")
  Reference_Genome<-"example.fasta"
}
# Save output in S4 Class object
return(new("Example_Data",
           reference=Reference_Genome,
           alternative=Alternative_Genome))}
```

5. Collection of potential errors and warning messages

R errors function based errors:

- *"Error: internal: buf != <newline>"*
  The Cause:
  This error is most commonly thrown by `qaSummary()` and `seeFastq()` in chapter 4, but may also occur when manually reading in a FASTQ file with `ShortRead::read.fastq()`.
  It is caused by a malformed FASTQ file where the number and width of the reads does not correspond with that of their quality scores. During pipeline testing it commonly occurred when using the generated example dataset when the random generator pulled unexpected values upon which the reads and quality scores are based.

The Solution:

If this error occurred when using example data, try generating a new dataset with the same code of chapter two. If the error persists, return all example data variables such as read length and reference genome lengths to the default settings.

If this error occurred when using ones own dataset, the solution is more complex. A corrupted FASTQ file may have been caused during downloading or uploading to and from the database. Another cause may be preprocessing outside of the R pipeline by external software. In this case, try to use the raw FASTQ files to test whether these were already malformed.

If the problem persists it is best to contact the authors of the FASTQ data, or the sequencing company to verify that the malformation was not caused at their end.

- *"running command '"C:/Users/Username/Documents/R/win-library/3.6/Rbowtie2/bowtie2-align-s" TRUE -x Example_Index -S example.sam -1 example_1.fastq -2 example_2.fastq' had status 1".*

  The Cause:

  This error states that R was unable to receive results from the Rbowtie2 aligner because it was not executed.

  The Solution:

  The Rbowtie package sends its commands to a package executable outside of R. Ensure that the entire Rbowtie package directory is whitelisted to prevent antivirus software from blocking the executable.

- *Cannot open file '…': No such file or directory*

  The Cause:

  This error is thrown when a function is unable to find the defined input file in the working directory.

  The Solution:

  Ensure that the correct working directory is defined by typing `getwd()` in the lower left "Console" panel. If the working directory is incorrect, define the correct path in chapter 1. If the working directory is correct, check that the defined input file name is spelled correctly, including the file extension. Also check that this file is indeed present in the defined working directory by typing `list.files(getwd())` into the R console.

- *During Package installation: Update all/some/none? [a/s/n]:*

  The Cause:

  This message is not an error or warning, but this question may cause confusion.

  R always checks the dependencies of the packages it is installing to ensure that these have already been installed and are up-to-date. If one or more of the dependencies are outdated, R will ask if one wants to update them.

  The Solution:

  It is good practice to keep all packages up-to-date, so simply update the packages by typing "a" (without ") into the lower left Console panel. R will then proceed with the installation.

- *Error in … Could not find function …*

  The Cause:

  R could not find the function called upon by the code.

  The Solution:

  Ensure that all required libraries are loaded before executing the code. The most common cause is accidentally deleting or forgetting installations.

  If the error persists, also ensure that the code is typed correctly and that `?FuntionName()` returns the help page of the offending function.

- *Error in … Object … not found*

  The Cause:

  R is unable to find the object named in the error within the workspace. The first part of the error further defines what function ran into the problem to pinpoint the line where the problem occurred.

  The Solution:

  First check whether the object is visible in the workspace by scrolling through the upper right panel "Environment". If the object is visible, please check that the spelling and capital letters are correct. If the object is not visible in the workspace, ensure that it has been defined by rerunning the input code of the respective chunk.
- *Error in library(…) : There is no package called …*

  The Cause:

  R could not find the library that one is trying to load with `library()` as it is not installed or does not exist.

  The Solution:

  Go to the down right tab "Packages" and search for the package to ensure it is not installed. If it is installed, click the small cross right of the package name to remove it from your computer. Next, reinstall the package and try to load the library again.
- *Error in save.image(Workspace_Name) : object 'Workspace_Name' not found*

  The Cause:

  Each chunk in the pipeline automatically saves its output in the workspace. If no workspace name is defined, however, it is unable to save its output.

  The Solution:

  Rerun the code in chapter one where both the working directory and workspace save name are defined. The code will now save its output to this file.

  The error can be ignored if one does not want to save its workspace during testing, but for any other situation it is strongly recommended to save the workspace anyway.
- *Error: Input/Output no input files found*

  The Cause:

  This error is thrown by qaSummary when it is unable to find the defined input file in the working directory.

  The Solution:

  Ensure that the correct working directory is defined by typing `getwd()` in the lower left "Console" panel. If the working directory is incorrect, define the correct path in chapter 1. If the working directory is correct, check that the defined input file name is spelled correctly, including the file extension. Also check that this file is indeed present in the defined working directory by typing `list.files(getwd())` into the R console.
- *In checkFileCreatable(paste0(basename, ".adapter1"), "file1", overwrite) :*
  For argument `file1`, file exist :C:/Users/UserName/Reads.adapter1. It will be overwrited

  The Cause:

  This is a warning and not an error. It simply states that the empty file generated by the code will be filled with the detected adapter sequences.

  The Solution:

  The warning has no effect on the analysis and is always outputted by RBowtie2. No solution is needed and the warning can be ignored.
- *In system(call, intern = TRUE, show.output.on.console = TRUE) :*
  running command '"C:/Users/UserName/Documents/R/win-library/3.6/Rbowtie2/AdapterRemoval" --file1 example_1.fastq --adapter1 --output1

C:/Users/UserName/TRIM_example_1.fastq --file2 example_2.fastq --adapter2 --output2 C:/Users/drema/Documents/TRIM_example_2.fastq --threads 3' had status 1

The Cause:

This long error is thrown by Rbowtie2 when attempting to remove adapters when none were detected in the previous chapter or when the adapter detection of the previous chapter was not performed.

The Solution:

The adapters cannot be removed as there are none to remove from the reads in the first place. The error is therefore solved by skipping the adapter removal step by changing Adapter_Removal=TRUE in the input chunk of Chapter 5 to Adapter_Removal=FALSE.

R Bash Terminal Related Errors

- *Rscript Variant_Calling_in_R_Alignment* or executing any other R script returns:
  In file.create cannot create file '…', reason Permission Denied. Execution Halted

  The Cause:

  The command to execute an R script calls upon Rscript.exe to perform this task. It may happen that Rscript.exe does not have the rights to operate on user files.

  The Solution:

  Set the directory of the terminal within the user account with the command *cd /c/Users/Username.* If the problem persists, go to the location of the Rscript.exe file (found by typing *which Rscript* into the terminal) and right click on the executable. Select properties, go to the Security tab and click on Edit. Note that this requires administration rights on the computer. Next, tick all the boxes under allow and click on OK. The Rscript executable should now have all rights needed to create and edit files.
- *RScript command not found*

  The Cause:

  This error tends to pop up when using Git Bash on Windows. The problem is that the command RScript is not found because it is not present in the $PATH, which is where Git Bash looks for potential commands.

  The Solution:

  In Windows, open the configuration panel and go to System>Advanced System Settings>Environment Variables. In the lower part of the window that opens, a list of system variables is given, including $PATH (simply named Path). Click on Path and select Edit, a new window opens listing all commands in Path. Click on New and fill in the path to the RScript executable. This path is similar to 'C:\Program Files\R\R-3.6.2\bin\x64' and may vary depending on the R version installed.

  Now select Okay on all Settings windows and restart Git Bash. The error should be solved.
- *setwd(opt$dir) : cannot change working directory*

  The Cause:

  When executing Rscript example.R an invalid working directory was submitted. This could be a map where the user does not have editing rights for, a non-existing directory path or an unrecognisable path such as $pwd.

  The Solution:

  Change the given working directory -d into a valid map. When pwd was used, try writing out the path instead.
- *Error in checkFileExist(seq1, "seq1") :*
  For argument `seq1`, file does not exist: `Example`

The Cause:

R cannot find the specified file in the working directory.

The Solution:

Check whether the specified file name was typed correctly and if the corresponding extension is defined as well. ‘Example’ may not work where ‘Example.fastq’ will since R uses full file names to locate the data. Also ensure that the specified file is present in the working directory.

- *Error in checkFileExist(paste0(bt2Index, ".rev.1.bt2"), "bt2Index") :*
  For argument `bt2Index`, file does not exist:

  The Cause:

  The Bowtie2 aligner uses reference index files for alignment. This error indicates that a part of this index is missing and that alignment therefore cannot be executed.

  The Solution:

  Remove all previously made index files and rerun the indexing step. Bowtie2 should then create a new and complete index.