Step 0: Check the Illumina encoding
In this step the encoding of the FastQ files will be checked. Older (2012 and older) sequence data contains the old Phred+64 encoding (this is called Illumina 1.5 encoding), new sequence data is encoded in Illumina 1.8 or 1.9 (Phred+33). If the data is 1.5, it will be converted to 1.9 encoding

Scriptname: CheckIlluminaEncoding
Input:. Fastq files

Step 1: Calculate QC metrics on raw data
In this step, Fastqc, quality control (QC) metrics are calculated for the raw sequencing data. This is done using the tool FastQC. This tool will run a series of tests on the input file. The output is a text file file containing the output data which is used to create a summary in the form of several HTML pages with graphs for each test. Both the text file and the HTML document provide a flag for each test: pass, warning or fail. This flag is based on criteria set by the makers of this tool. Warnings or even failures do not necessarily mean that there is a problem with the data, only that it is unusual compared to the used criteria. It is possible that the biological nature of the sample means that this particular bias is to be expected.

Toolname: FastQC
Scriptname: Fastqc
Input: raw sequence file in the form of a gzipped fastq file (.fq.gz)
Output: fastqc.zip archive containing amongst others the HTML document and the text file

Step 2: Read alignment against reference sequence
In this step, the Burrows-Wheeler Aligner (BWA) is used to align the (mostly paired end) sequencing data to the reference genome. The method that is used is BWA mem. The output is a SAM file.

Scriptname: BwaAlign
Input: raw sequence file in the form of a gzipped fastq file (.fq.gz)
Output: SAM formatted file (.sam)

Figure 1: Workflow of the inhouse sequence analysis pipeline.
Toolname: BWA

Step 3: Convert SAM to BAM
Using the Picard SamFormatConverter the SAM file generated by the previous step is converted to a compressed binary format (BAM).

Toolname: Picard SamFormatConverter
Scriptname: SamToBam
Input: SAM file generated in step 2
Output: compressed binary (BAM) format (.bam)

Step 4: Sort BAM and build index
The reads in the BAM file are sorted (coordinate based) using Picard SortSam and an index file is generated using Picard BuildBamIndex. The index file (with extension .bam.bai) allows for efficient random access to the BAM file and is used by many tools to speed up reading from a BAM file.

Toolname: Picard SortSam and BuildBamIndex
Scriptname: SamSort
Input: BAM file from step 5
Output: sorted BAM file (.sorted.bam) and matching index file (.sorted.bam.bai)

Step 5: Merge BAMs and build index
To improve the coverage of sequence alignments, a sample can be sequenced on multiple lanes and/or flowcells. If this is the case for the sample(s) being analyzed, this step merges all BAM files of one sample and indexes this new file. If there is just one BAM file for a sample, nothing happens.

Toolname: Picard MergeSamFiles and BuildBamIndex
Scriptname: MergeBam
Input: BAM files from step 4
Output: merged BAM file (.merged.bam)

Step 6: Mark duplicates
In this step, the BAM file is examined to locate duplicate reads, using Picard MarkDuplicates. A mapped read is considered to be duplicate if the start and end base of two or more reads are located at the same chromosomal position in comparison to the reference genome. For paired-end data the start and end locations for both ends need to be the same to be called duplicate. One read pair of these duplicates is kept, the remaining ones are flagged as being duplicate. A BAM file with the flags and a new index (BuildBamIndex) are generated in this step. Note that no reads are discarded during this step.

Toolname: Picard MarkDuplicates and BuildBamIndex
Scriptname: Markduplicates
Input: sorted BAM file from generated in step 6
Output: BAM file with duplicates flagged (.dedup.bam), deduplication metrics (.dedup.metrics), BAM index file (.dedup.bam.bai)

Step 7: Indel calling with Pindel
In this step, the GATK IndelGenotyper calls indels from the merged BAM file. The indels are written to a VCF file, along with information such as difference in length between REF and ALT alleles, type of structural variant end information about allele depth.

Toolname: Pindel
Scriptname: Pindel
Input: merged BAM file
Output: indels in VCF (.pindel.calls.merged.vcf).

Step 8: Realign intervals
The GATK tool IndelRealigner realigns reads at the suspicious regions found in step 9. It takes the deduplicated BAM file and the list of suspicious intervals created at step 9 as input.

Toolname: GATK (3.3-0-g37228af) IndelRealigner
Scriptname: IndelRealignment
Input: BAM file with duplicates flagged and list of suspicious intervals
Output: realigned BAM file (.realigned.bam)

Step 9 and 10: Recalibrate alignment base quality scores
This step recalibrates the base quality scores of each read in the alignment. This is done in two steps: first a number of covariates are collected for each base and next this information is used to recalibrate the base quality scores. Recalibration is done for all bases except those in dbSNP 129.

1. The GATK CountCovariates walks over each read in the matefixed BAM file and tabulates data of every combination of the used covariates. For each such combination (or bin) the number of bases conforming to that bin is counted, as well as how often these bases mismatch the reference genome (known sites in dbSNP excluded). From this an empirical base quality score is derived. The output is a comma separated file containing each bin including the empirical quality score. The covariates currently used are read group, base quality score, machine cycle producing this base and dinucleotide (current base and previous base)

2. Based on the bins, the empirical quality scores, the actual quality scores and covariates machine cycle and dinucleotide, the GATK TableRecalibration calculates a new quality score for each base

Toolname: GATK (3.3-0-g37228af) BaseRecalibrator
Scriptname: GenerateCoveriateTablesBefore
Input: matefixed BAM
Output: comma separated covariates table (.before.recal.table)

Toolname: GATK (3.3-0-g37228af) PrintReads
Scriptname: QualityScoreRecalibration
Input: matefixed BAM and covariates table
Output: recalibrated BAM file (.merged.dedup.realigned.bqsr.bam)

Step 11: Count covariates after recalibration
Repetition of step 9. In step 12, both covariate tables are analyzed, making it possible to compare.

Toolname: GATK (3.3-0-g37228af) BaseRecalibrator
Scriptname: GenerateCovariateTablesPost
Input: recalibrated BAM
Output: comma separated covariates table (.post.recal.table)

Step 12: Analyze covariates before and after realignment
Using the GATK AnalyzeCovariates both covariates tables are analyzed to produce graphs. For each table a directory is created, containing some data files and some PDF files. These PDF files graphically show the various metrics and characteristics of the reported quality scores.

Toolname: GATK (3.3-0-g37228af) AnalyzeCovariates
Scriptname: AnalyzeQualityScoreRecalibration
Input: covariates tables from before and after recalibration
Output: graphs in pdf and csv format (.merged.dedup.realigned.bqsr.csv)

Step 13: Calculate alignment QC metrics
In this step, QC metrics are calculated for the alignment created in the previous steps. This is done using several QC related Picard tools:

- CollectAlignmentSummaryMetrics
- CollectGcBiasMetrics
- CollectInsertSizeMetrics
- MeanQualityByCycle (machine cycle)
- QualityScoreDistribution
- CalculateHsMetrics (hybrid selection)
- BamIndexStats

These metrics are later used to create tables and graphs (step 29). The Picard tools also output a PDF version of the data themselves, containing graphs.

Toolname: several Picard QC tools
Scriptname: CollectBamMetrics
Input: recalibrated BAM file (.merged.dedup.realigned.bqsr.bam)
Output: alignmentmetrics, gcbiasmetrics, insertsizemetrics, meanqualitybycycle, qualityscoredistribution, hsmetrics, bamindexstats (text files and matching PDF files)

Step 14: Call variants (VariantCalling)
The GATK HaplotypeCaller estimates the most likely genotypes and allele frequencies in an alignment using a Bayesian likelihood model. This leads to a posterior probability of a variant

allele at a site. SNPs are written to a VCF file, along with information such as genotype quality, allele frequency, strand bias and read depth for that SNP.

Toolname: GATK (3.3-0-g37228af) HaplotypeCaller
Scriptname: UnifiedGenotyper (see: GCC_P0009_Pipeline_tools_and_resources.docx)
Input: merged BAM file
Output: VCF file with SNPs (.chr${chr}.variant.calls.vcf)

Step 15: Merge chromosomes and split variants
Running vcftools (*concat*) to merge all the files created in the VariantCalling step (Step 14) into one. Then use vcftools to split the indels and snps and create 2 new files.

Tools: vcftools (0.1.12a)
Scriptname: MergeChrAndSplitVariants
Input: files created in step 14 (.chr${c}.variant.calls.vcf)
Output: merged snp file (.SNPs.calls.GATK.merged.vcf) and merged indel file (.indels.calls.GATK.merged.vcf)

Step 16: VariantFiltration
Based on certain quality thresholds (based on GATK best practices) the SNPs are filtered, and marked as Lowqual or Pass.

Toolname: GATK (3.3-0-g37228af) VariantFiltration
Scriptname: VariantFiltration
Input: merged snp file from step 15 (.SNPs.calls.GATK.merged.vcf)
Output: filtered vcf file (.SNPs.calls.GATK.merged.filtered.vcf)

Step 17: IndelMergeVcf
Run GATK SelectVariants and restrict the output to a set of intervals. Merge GATK and Pindel output into one file with bcfTools. Afterwards making new header and replace the header in the vcf using Tabix. The new vcf file will only contain the Indels called by both GATK and Pindel.

Toolname:- GATK (3.3-0-g37228af) SelectVariants
- BcfTools (0.2.0) merge
- Tabix (0.2.6)

Input: GATK output from step step 15 (.indels.calls.GATK.merged.vcf) and Pindel output (.output.pindel.merged.vcf)
Output: Vcf format file (.indels.calls.GATK.Pindel.merged.vcf)

Step 18: (a. Indel and b. SNP) SnpEff
SnpEff is a variant annotation and effect prediction tool. It annotates and predicts the effects of variants on genes (such as amino acid changes)

Toolname: SnpEff (3.6c) eff
Scriptname: 18a SnpEffIndel; 18b SnpEffSNP
Input: 18a: predicted variants calculated in step 17 (.indels.calls.GATK.Pindel.merged.vcf)
18b: predicted variants calculated in step 16 (.SNPs.calls.GATK.merged.filtered.vcf)
Output: 18a: (.indels.calls.GATK.Pindel.merged.snpEff.vcf
18b: (.SNPs.calls.GATK.merged.filtered.snpEff.vcf)

Step 19a: VariantAnnator Indels
In this step, two tools are used to functionally annotate the SVs. First, snpEff predicts the effect of SVs on genes and outputs an intermediate VCF file with this new annotation added. Additionaly, an HTML file with some statistics and a text file with SNPs per gene and region are produced. Secondly, the GATK VariantAnnotator includes the intermediate snpEff annotation to the VCF file generated in step 24. Annotations are:

SNPEFF_EFFECT - The highest-impact effect resulting from the current variant (or one of the highest-impact effects, if there is a tie)

SNPEFF_IMPACT - Impact of the highest-impact effect resulting from the current variant (HIGH, MODERATE, LOW, or MODIFIER)
SNPEFF_FUNCTIONAL_CLASS - Functional class of the highest-impact effect resulting from the current variant (NONE, SILENT, MISSENSE, or NONSENSE)
SNPEFF_CODON_CHANGE - Old/New codon for the highest-impact effect resulting from the current variant
SNPEFF_AMINO_ACID_CHANGE - Old/New amino acid for the highest-impact effect resulting from the current variant
SNPEFF_GENE_NAME - Gene name for the highest-impact effect resulting from the current variant
SNPEFF_GENE_BIOTYPE - Gene biotype for the highest-impact effect resulting from the current variant
SNPEFF_TRANSCRIPT_ID - Transcript ID for the highest-impact effect resulting from the current variant
SNPEFF_EXON_ID - Exon ID for the highest-impact effect resulting from the current variant
(source: http://www.broadinstitute.org/gatk/guide/article?id=50)

Toolname: snpEff (3.6c) snpEff and GATK (3.3-0-g37228af) VariantAnnotator
Scriptname: StructuralVariantAnnotator
Input: VCF file (.indels.calls.GATK.Pindel.merged.vcf)
Output: SNPeff VCF file (.snpEff.annotated.indels.final.vcf)

Step 19b: VariantAnnator SNPs
19a In this step, two tools are used to functionally annotate the SNPs. First, snpEff predicts the effect of SNPs on genes and outputs an intermediate VCF file with this new annotation added. Additionaly, an HTML file with some statistics and a text file with SNPs per gene and region are produced.
Secondly, the GATK VariantAnnotator calculates annotations using the called SNPs and the alignment, and also includes the intermediate snpEff file. The output is a VCF file with functionally annotated SNPs. Some example annotations are:

- allele counts
- fraction of reads that contain a spanning (completely covering) deletion at this site
- number of reads that aligned with mapping quality zero at this site
- read depth, both total and reads carrying the alternative and reference base at this site
- strand bias
- largest homopolymer run of the variant allele in either direction on the reference
- GC content of the reference within 50 bp in either direction of this site

Toolname: SnpEff (3.3-0-g37228af) and GATK (3.3-0-g37228af) VariantAnnotator
Scriptname: VariantAnnotator
Input: SNPs VCF file (.SNPs.calls.GATK.merged.filtered.vcf)
Output: SNPeff VCF file (.snpEff.annotated.snps.vcf)

Step 20: dbNSFP Annotation
dbNSFP is a database developed for functional prediction and annotation of all potential non-synonymous single-nucleotide variants (nsSNVs) in the human genome
Annotations are:
Ensembl_geneid,GERP++_RS,
Polyphen2_HDIV_pred
Polyphen2_HVAR_pred
SIFT_score,CADD_raw
CADD_raw_rankscore
CADD_phred
FATHMM_score
SiPhy_29way_logOdds
phastCons100way_vertebrate

1000Gp1_EUR_AF
ESP6500_EA_AF

Toolname: SnpEff (3.6c) SnpSift dbnsfp
Scriptname: dbNSFPAnnotation
Input: .snpEff.annotated.snps.vcf
Output: annotated vcf file (.snpEff.annotated.snps.dbnsfp.final.vcf)

Step 21a: Convert structural variants VCF to table
In this step the indels in VCF format are converted into a tabular format using Perlscript vcf2tab by F. Van Dijk (see: GCC_P0009_Pipeline_tools_and_resources.docx).

Toolname: vcf2tab.pl
Scriptname: IndelVcfToTable
Input: (.snpEff.annotated.indels.final.vcf)
Output: (.snpEff.annotated.indels.final.vcf.table)

Step 21b: Convert SNPs VCF to table
In this step the SNPs in VCF format are converted into a tabular format using Perlscript vcf2tab by F. Van Dijk (see: GCC_P0009_Pipeline_tools_and_resources.docx).

Toolname: vcf2tab.pl
Scriptname: SNPVcfToTable
Input: (.snpEff.annotated.snps.dbnsfp.final.vcf)
Output: (.snpEff.annotated.snps.dbnsfp.final.vcf.table)

Step 22: Concordance Check

Scriptname: ConcordanceCheck
As a last measure of quality of the SNPs reported in the previous steps, the concordance between the SNPs and the SNPs on a SNP array of the same sample is checked. If at least 1,000 SNPs overlap and the concordance is around 97% or higher, the SNPs reported in the previous steps are accepted as being of reliable quality and exclude a potential sample swap. The concordance check is done in several steps (provided GenomeStudio final report is present):

- sed, awk, uniq, sort, grep to convert the GenomeStudio final report of an analyzed SNP-array (see: GCC_0003_Generate_finalReport_from_GenomeStudio.docx) into map, lgen and fam files (plink long-format fileset)
- plink-1.07 recode which is used for creating a *.bed file
- plink 1.08 unpublished development version recode which is used to create a genotype in *.vcf format
- command line Perl to change the header
- sed and awk to convert from vcf to bed
- fastaFromBed from BEDTools-Version-2.11.2 to create a uscs style tab delimited fasta file from the bed file, using reference sequence build 36
- align-vcf-to-ref.pl from inhouse_scripts
- liftOverVCF.pl from GATK-1.0.5069 which is used for lifting hg18 VCF files to hg19.
- head, sed, cat to change header
- iChip_pos_to_interval_list.pl from inhouse_scripts is used to create an interval list of array SNPs which is used to call inhouse SNPs
- GATK-1.2-1-g33967a4 UnifiedGenotyper which is used for calling SNPs on all positions known to be on array and VCF and calculating the concordance between array SNPs and inhouse pipeline SNPs
- change_vcf_filter.pl from inhouse scripts is used to change the FILTER column from GATK "called SNPs". All SNPs having Q20 & DP10 change to "PASS", all other SNPs are "filtered" (not used in concordance check)

- GATK-1.2-1-g33967a4 VariantEval to calculate condordance between genotype SNPs and GATK "called SNPs"
- echo to prepare the header of the output concordance file
- R script extract_info_GATK_variantEval_V3.R (using library from GATK-1.3-24-gc8b1c92) from inhouse scripts to format the concordance output file

When the final report from GenomeStudio is not present (there is no array file) and no concordance can be calculated, an empty concordance output file with a header and one row of NAs is created on the fly.

Toolname: BASH programs (sed, awk, uniq, sort, grep, head, cat, echo), plink-1.07, plink 1.08 (unpublished development version), fastaFromBed (BEDTools-Version-2.11.2), align-vcf-to-ref.pl from inhouse_scripts, liftOverVCF.pl from GATK-1.0.5069, iChip_pos_to_interval_list.pl from inhouse_scripts, GATK-1.2-1-g33967a4 UnifiedGenotyper, change_vcf_filter.pl from inhouse scripts, GATK-1.2-1-g33967a4 VariantEval, extract_info_GATK_variantEval_V3.R (using library from GATK-1.3-24-gc8b1c92) from inhouse scripts
Scriptname: ConcordanceCheck
Input: SNP array file generated by GenomeStudio (_FinalReport.txt)
Output: concordance file (.concordance.ngsVSarray.txt)

Step 23: Generate quality control report
The step in the inhouse sequence analysis pipeline is to collect the statistics and metrics from each step that produced such data. From these, tables and graphs are produced. Reports are then created (using MarkDown) and written to a separate quality control (QC) directory, located IN RunNr/Results/qc/statistics (see: SOP_0001_data_structure.txt, Figure 2).

Using the R script getStatistics.sh (M.Dijkstra, see: SOP_0009_Pipeline_tools_and_resources.docx), the following metrics are collected:
- hybrid selection metrics after merging
- alignment summary metrics after merging
- insert size metrics after merging
- concordance

These are used for the main QC report. Information on deduplication (step 6) is reported in a separate file using the R script getDedupInfo.sh. For each sample, SNP statistics are collected and reported as well. Additionally, a figure of the inhouse sequence analysis pipeline (same as figure 1) is produced in the QC directory. Finally, the quality control report MarkDown file is generated.

Toolname: getStatistics.R, getDedupInfo.R, createSNPTable.R
Scriptname: QCReport
Input: *.hsmetrics, *.alignmentmetrics, *.insertsizemetrics and *.dedup.metrics.concordance.ngsVSarray.txt
Output: A quality control report MarkDown(*_QCReport.md)

Step 24:Prepare data to ship to the customer
In this last step the final results of the inhouse sequence analysis pipeline are gathered and prepared to be shipped to the customer. The pipeline tools and scripts write intermediate results to a temporary directory (see: GCC_P0001_data_structure.docx). From these, a selection is copied to a results directory. This directory has five subdirectories:

- `alignment`: the merged BAM file with index
- `coverage`: coverage statistics and plots
- `coverage_visualization`: coverage BEDfiles
- `qc`: all QC files, from which the QC report is made
- `rawdata/ngs`: symbolic links to the raw sequence files and their md5 sum
- `snps`: all SNP calls in VCF format and in tab-delimited format
- `structural_variants`: all SVs calls in VCF and in tab-delimited format

Additionally, the results directory contains the final QC report, the worksheet which was the basis for this analysis (see 4.2) and a zipped archive with the data that will be shipped to the client (see: GCC_P0006_Datashipment.docx). The archive is accompanied by an md5 sum and README file explaining the contents.

Scriptname: CopyToResultsDir
See also: GCC_0001_data_structure.docx and GCC_P0006_Datashipment.docx

Step 25: Check if all files are finished

This step is checking if all the steps in the pipeline are actually finished. It sometimes happens that a job is not submitted to the scheduler. If everything is finished than it will write a file called CountAllFinishedFiles_CORRECT, if not it will make CountAllFinishedFiles_INCORRECT. When it is not all finished it will show in the CountAllFinishedFiles_INCORRECT file which files are not finished yet.

Scriptname: CountAllFinishedFiles
Input: all .sh scripts + all .sh.finished files in the jobs folder
Output: CountAllFinishedFiles_CORRECT or CountAllFinishedFiles_INCORRECT