Application Note

CLC Server Command Line Tools: a powerful choice to optimize your analyses

Francesco Lescai, Winnie Ridderberg, Leif Schauser, Jonathan Jacobs

QIAGEN® Bioinformatics – Aarhus, Denmark

Introduction

CLC Command Line Tools is a client interface for CLC enterprise products, allowing users to carry out server-based analyses, workflow execution, data import and export, and various other data operations via the command line. These commands can then be included within wrapper scripts written in bash, python, Perl, R, and many others. The CLC Command Line Tools act as a client to the CLC Genomics Server, just like the CLC Genomics Workbench which is the graphical software user interface client more commonly used with CLC Servers.

The CLC Command Line Tools provide flexible access to CLC Server commands and workflows enabling scalable enterprise-wide solutions. The command line to CLC applications can be integrated into a wide range of scripting languages common in Biology and Bioinformatics.

In this Application Note we will cover four different use case examples, as summarized in the following Table:

Use-Case Examples	
1. Optimizing seed length in taxonomic classification of microbial communities	How to loop over different seed lengths in order to optimize taxonomic profiling launching one single tool
2. Optimizing variant calling settings for CRISPR analysis	How to loop over different probability thresholds, in order to optimize the highly sensitive identification of CRISPR-induced mutations in an NGS read mapping, using a simple workflow
3. Running the command line tool from any programming language	How to embed a call to CLC Server Command Line Tools in a script written in any programming language
4. Optimizing variant filtering parameters to improve accuracy	How to use a number of nested loops, in order to run multiple combinations of selected filtering parameters, and optimise a complex workflow to filter genetic variants

Running CLC Server Command Line Tools

Like the CLC Genomics Workbench client, the CLC Command Line Tools need to be installed on the computer the user wants to execute them from. Installer files can be downloaded for all operating systems from the web page **www.qiagenbioinformatics.com/products/clc-server-command-line-tools/**.

Once the installation is complete, the user can generate a token to access the CLC Genomics Server without writing the password in clear, by using the command:

clcserverkeystore -g

This will prompt a request for the password, which will remain hidden, and generate a token the user can pass to the command line in the following steps. A key difference to the CLC Genomics Workbench client is that the Command Line Tools need to be run interactively in order to execute a process on the Server. For this reason, when the user needs to run tools or workflows with a long execution time, we suggest to run them in background (either using a screen or nohup) on a computer that remains connected to the Server.

Executing commands is straightforward, requiring only some familiarity with the terminal and command line.

The user can quickly visualize the tools and workflows available on the server, by typing the commands:

clcserver \

-S yourserver \

-U youruser \

-W XXX-PASS-TOKEN-AAAA-BBB \

[-G your_grid_queue]

The list of commands will be different depending on the choice of the -G option that enables running jobs on a computing grid. Depending on the server configuration, some tools or workflows might be available only for execution in a grid environment[1].

Generally, tools and workflows might require input files or destination folders referring to a specific location: a consistent way of formatting the location path is represented in Figure 1.

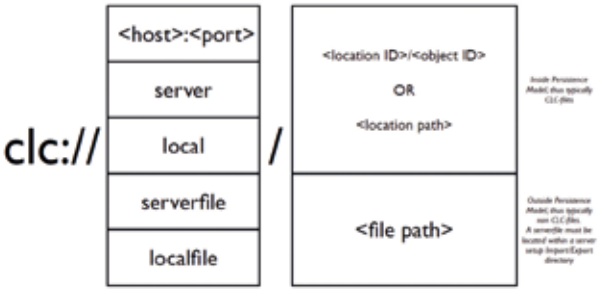

Figure 1. Format of CLC locations path

Should the user be in doubt about the location path, each location that is configured on the Server can be visualized on the CLC Genomics Server web interface: under the tab "Element Info", both the human readable path, as well as the CLC object location can be visualized and copied to be used in the command line, as shown in Figure 2.

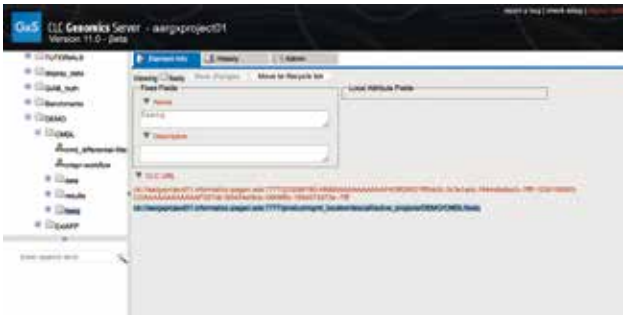

Figure 2. Element Info tab on the server, showing the CLC location path for any configured location.

[1] The complete list of commands is available at http://resources.qiagenbioinformatics.com/manuals/clcservercommandlinetools/current/index.php?manual=Usage_CLC_Genomics_Server.html

Use Case Example 1: Optimizing seed length in taxonomic classification of microbial communities

Taxonomic profiling of whole shotgun metagenomic NGS data can be based on mapping reads to a database of reference genomes. A single parameter is customizable for the read mapping – the minimum seed length. The seed length defines the minimum length for a match against the reference. If no match of this length against the reference can be found, the read is considered unmapped. Increasing the seed length will increase precision of results with fewer hits, but also potentially more false negatives. Lowering the seed length will increase the recall but at the cost of more potentially false positive results. Adjusting the minimum seed length can therefore be used to finding the optimal balance between precision and recall for a given dataset.

In the current example, we have used sequencing data from a standardized mock community provided by the NIH Human Microbiome Project (SRR172902). The mock community consists of 21 bacterial species in even mixture.

The taxonomic profiler identified all 21 bacterial species regardless of the minimum seed length. However, the number of false positive results decreased with increasing minimum seed length, reducing the number of false positive predictions from 19 at the lowest seed length of 22 to 11 at the higher seed lengths of 38 and 46. Iterating over the minimum seed length (Table 2) reveals the benefits of using the command line tools within an iteration loop to efficiently explore a range of parameters and optimize a workflow for a specific use case.

Table 2: Taxonomic classifications of a mock community in response to changing the seed length.

Minimum seed length [nt]	No. reads matching reference database	No. unclassified reads	Classifications at family level	Classifications at genus level	Classifications at species level	True positive detections at species level	False positive detections at species level
22	5,931,034	621,526	21	21	40	21	19
30	5,600,827	951,733	18	18	33	21	12
38	5,165,921	1,386,639	18	18	32	21	11
46	4,598,360	1,954,200	18	18	32	21	11

An example of the code necessary to run a single tool, in order to obtain the results shown above can be found in Code Box 1. We have scripted a simple bash loop on the value of an array, which contains the different seed values we wanted to test.

Code Box 1: Running the Taxonomic Profiling tool

```
seedArray=( 22 30 38 46 )
for seed in "${seedArray[@]}"
do
echo "executing taxo profile with seed value ${seed}" >>taxo.log

### creating a folder for each group of results
clcserver \
-S yourserver \
-U youruser \
-W XXX-PASS-TOKEN-AAAA-BBB \
-A mkdir \
-n Results_seed${seed} \
-t clc://yourserlver:7777/your_location/file/microbial

### launching the tool and setting destination to newly created folder
clcserver \
-S yourserver \
-U youruser \
-W XXX-PASS-TOKEN-AAAA-BBB \
-G your_grid_queue \
-A taxonomic_profiling \
-d clc://yourserlver:7777/your_location/file/microbial/Results_seed${seed} \
–references clc://yourserlver:7777/your_location/file/microbial/Microbial_genome_database \
-i clc://yourserlver:7777/your_location/file/microbial/HMP_even_illum_SRR172902_trimmed \
–min-seed-length ${seed}
done
```

The example in Code Box 1 also shows how the user can combine several commands in the same bash script; in this case, the chosen tool does not create a new folder to save the results in. Thus, we have added a preliminary step in the code to create a new folder, and then we have used it as destination for the results. With this choice, we can clearly identify the results obtained by running different parameters.

Use Case Example 2: optimizing variant calling settings for CRISPR analysis

The use of the CLC Command Line Tools is not limited to executing single tools, but can be extended to the workflow system available in the CLC Genomics Workbench and CLC Genomics Server environments. In order to view the workflows installed on the server, the user can use the command "-A list_workflows": the name of the installed workflow might be different from the one visible from the Workbench.

In this use case example, we analysed data from an experiment where the authors wished to test the efficiency of different variants of cas9 (nuclease and nickases)[1]. We designed an example workflow (Figure 3) for trimming NGS reads, map them to the reference, realign the reads and finally call the variants using the Low Frequency Variant Detection caller.

The aim was to test the effect of using different thresholds in the probability of the variant calling, and assess how this parameter change affects which mutations are called.

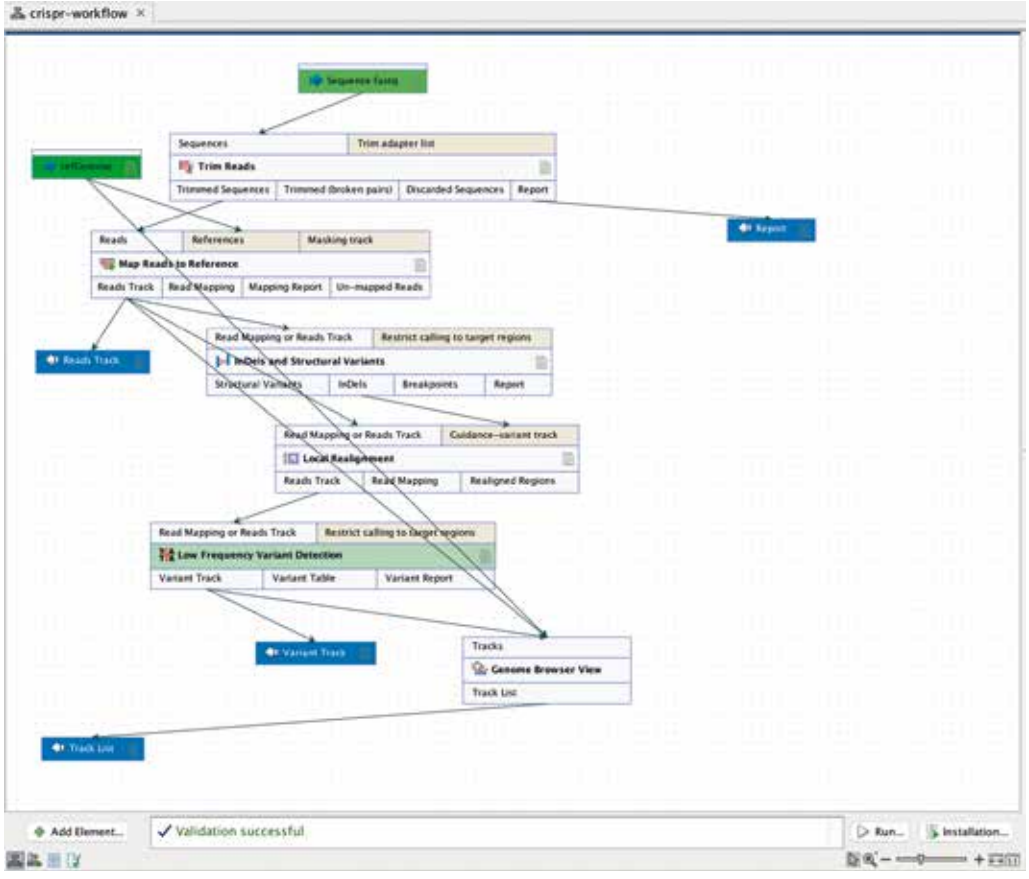

Figure 3. Example workflow to identify CRISPR-induced mutations

Code Box 2 shows the syntax of the code in Bash, used to run the workflow from the command line. Executing a workflow is just as easy as the execution of a single tool. During the design of the workflow, all parameters configured as "open" in the workflow settings will appear in the help message of the command line, and the requested input will be described. A default can also be defined, for the values of each of the open parameters.

Code Box 2: Optimizing calling significance threshold in Bash

```
sigArray=( 5 10 15 20 25 30 35 40 45 50 )
for sigValue in "${sigArray[@]}"
do
echo "executing sig = ${sigValue}" >>crispr_loop.log
clcserver \
-S yourserver \
-U youruser \
-W XXX-PASS-TOKEN-AAAA-BBB \
-A wf-company-workflow-name \
-G your_grid_queue \
–sequence-fastq-workflow-input clc://yourserIver:7777/your_location/file/CRSPR/110413_6_JY747_GCCAAT_L001_R1_001 \
-d clc://yourserIver:7777/your_location/file/CRSPR \
–low-frequency-variant-detection-required-significance ${sigValue} \
–reads-track-custom-output-name /sig${sigValue}/mapping_sig${sigValue} \
–report-custom-output-name /sig${sigValue}/report_sig${sigValue} \
–track-list-custom-output-name /sig${sigValue}/browser_sig${sigValue} \
–variant-track-custom-output-name /sig${sigValue}/variants_sig${sigValue}
done
```

In the workflow shown in Figure 3, we also created a genome view (Track List), which allows easy inspection of the results. Figure 4 shows the effect of changing the significance parameters, and we can quickly observe how increasing the significance threshold leads to the identification of a larger number of CRISPR-induced variants.

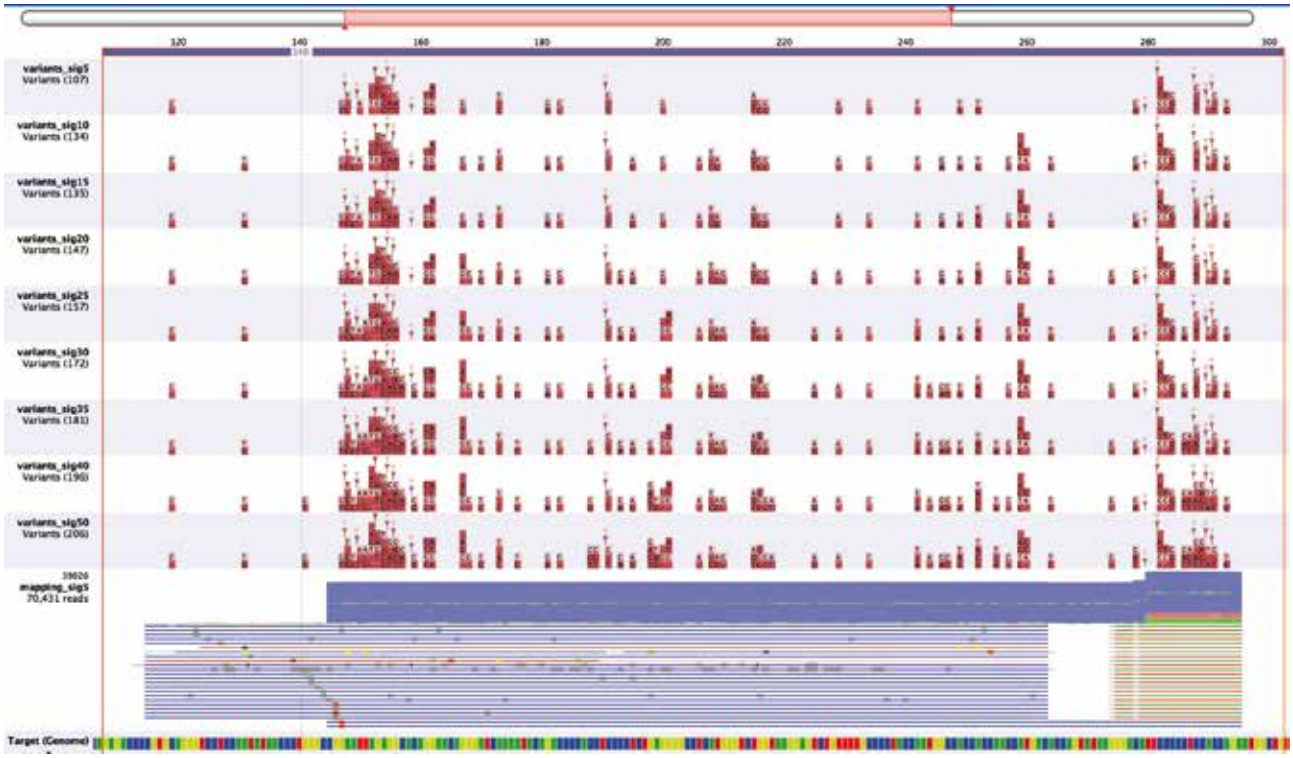

Figure 4. Effect of different significance thresholds in the number of variants identified

Use Case Example 3: running the command line tool from any programming language

The CLC Server Command Line Tools invokes a system command. For this reason, the users are not bound to use only Bash, but have the flexibility to launch a system command from any programming language available.

In the following boxes we used the same code presented in the previous Use Case Example 2, and translated it from Bash into Python (Code Box 3), Perl (Code Box 4) and R (Code Box 5).

Code Box 3: Embedding a call to CLC Command Line Tools in Python

```
#!/usr/bin/python

import os

input = "clc://yourserlver:7777/your_location/file/CRSPR/110413_6_JY747_GCCAAT_L001_R1_001"
destination = "clc://yourserlver:7777/your_location/file/CRSPR/"

params = [25, 50]

for param in params:
        outreads = "/PY/sig" + str(param) + "/mapping_sig" + str(param)
        outname = "/PY/sig" + str(param) + "/report_sig" + str(param)
        outrack = "/PY/sig" + str(param) + "/browser_sig" + str(param)
        vartrack = "/PY/sig" + str(param) + "/variants_sig" + str(param)
        print("executing on param = " + str(param))
        print("outreads = " + outreads)
        command = clcserver \
        -S yourserver \
        -U youruser \
        -W XXX-PASS-TOKEN-AAAA-BBB \
        -G your_grid_queue -A wf-company-workflow-name \
        –sequence-fastq-workflow-input " + input + " -d " + destination + " \
        –low-frequency-variant-detection-required-significance " + str(param) + " \
        –reads-track-custom-output-name " + outreads + " \
        –report-custom-output-name " + outname + " \
        –track-list-custom-output-name " + outrack + " \
        –variant-track-custom-output-name " + vartrack
        print(command)
        os.system(commands
```

Code Box 4: Embedding a call to CLC Command Line Tools in Perl

```
#!/usr/bin/perl
use strict;
use warnings;

my $input = "clc://yourserlver:7777/your_location/file/CRSPR/110413_6_JY747_GCCAAT_L001_R1_001";
my $destination = "clc://yourserlver:7777/your_location/file/CRSPR/";

my @params = (25, 50);

foreach my $param (@params){

        my $outreads = "/PY/sig$param/mapping_sig$param";
        my $outname = "/PY/sig$param/report_sig$param";
        my $outrack = "/PY/sig$param/browser_sig$param";
        my $vartrack = "/PY/sig$param/variants_sig$param";
        print STDERR "executing on param = $param\n";
        print STDERR "outreads = $outreads\n";
        my $command = "clcserver \\
                        -S yourserlver \\
                        -U youruser \\
                        -W XXX-PASS-TOKEN-AAAA-BBB  \\
                        -G your_grid_queue -A crispr-workflow \\
                        –sequence-fastq-workflow-input $input -d $destination \\
                        –low-frequency-variant-detection-required-significance $param \\
                        –reads-track-custom-output-name $outreads \\
                        –report-custom-output-name $outname \\
                        –track-list-custom-output-name $outrack \\
                        –variant-track-custom-output-name $vartrack";
        print STDERR $command."\n";
        system($command);
}
```

Code Box 5: Embedding a call to CLC Command Line Tools in R

```
#!/usr/bin/Rscript

input = "clc://yourserlver:7777/your_location/file/CRSPR/110413_6_JY747_GCCAAT_L001_R1_001"

destination = "clc://yourserlver:7777/your_location/file/CRSPR/"

params = c(25,50)

for (param in params){
  outreads = paste0("/PY/sig", param, "/mapping_sig", param)
  outname = paste0("/PY/sig", param, "/report_sig", param)
  outrack = paste0("/PY/sig", param, "/browser_sig", param)
  vartrack = paste0("/PY/sig", param, "/variants_sig", param)

  writeLines(paste0("executing on param ", param))
  writeLines(paste0("outreads = ", outreads))

  command = paste0("clcserver ",
            "-S yourserlver ",
            "-U youruser ",
            "-W XXX-PASS-TOKEN-AAAA-BBB  ",
            "-G your_grid_queue -A crispr-workflow ",
            "–sequence-fastq-workflow-input ", input, " -d ", destination,
            " –low-frequency-variant-detection-required-significance ", param,
            " –reads-track-custom-output-name ", outreads,
            " –report-custom-output-name ", outname,
            " –track-list-custom-output-name ", outrack,
            " –variant-track-custom-output-name ", vartrack)
  writeLines(command)
  system(command)
}
```

These examples show the wide range of opportunities to integrate CLC Server Command Line tools.

Use Case Example 4: optimizing variant filtering parameters to improve accuracy

The flexibility of the CLC Server Command Line Tools allows users to expand their applications to complex workflows and use the command line to perform parameter sweeps.

In order to show a more complex example, we have designed a workflow (Figure 5) that accepts the variants called from the exome derived NGS sequences of the NA12878 DNA standard sample as input, and differentially filters SNVs and INDELs as a function of three parameters for each variant type.

In order to assess the effect of the large number of combinations of those six parameters, at each execution the workflow compares the results with the validated set of variants from Genome in a Bottle in high-confidence regions, generating tracks for the false positives (FPs), false negatives (FNs) and true positives (TPs).

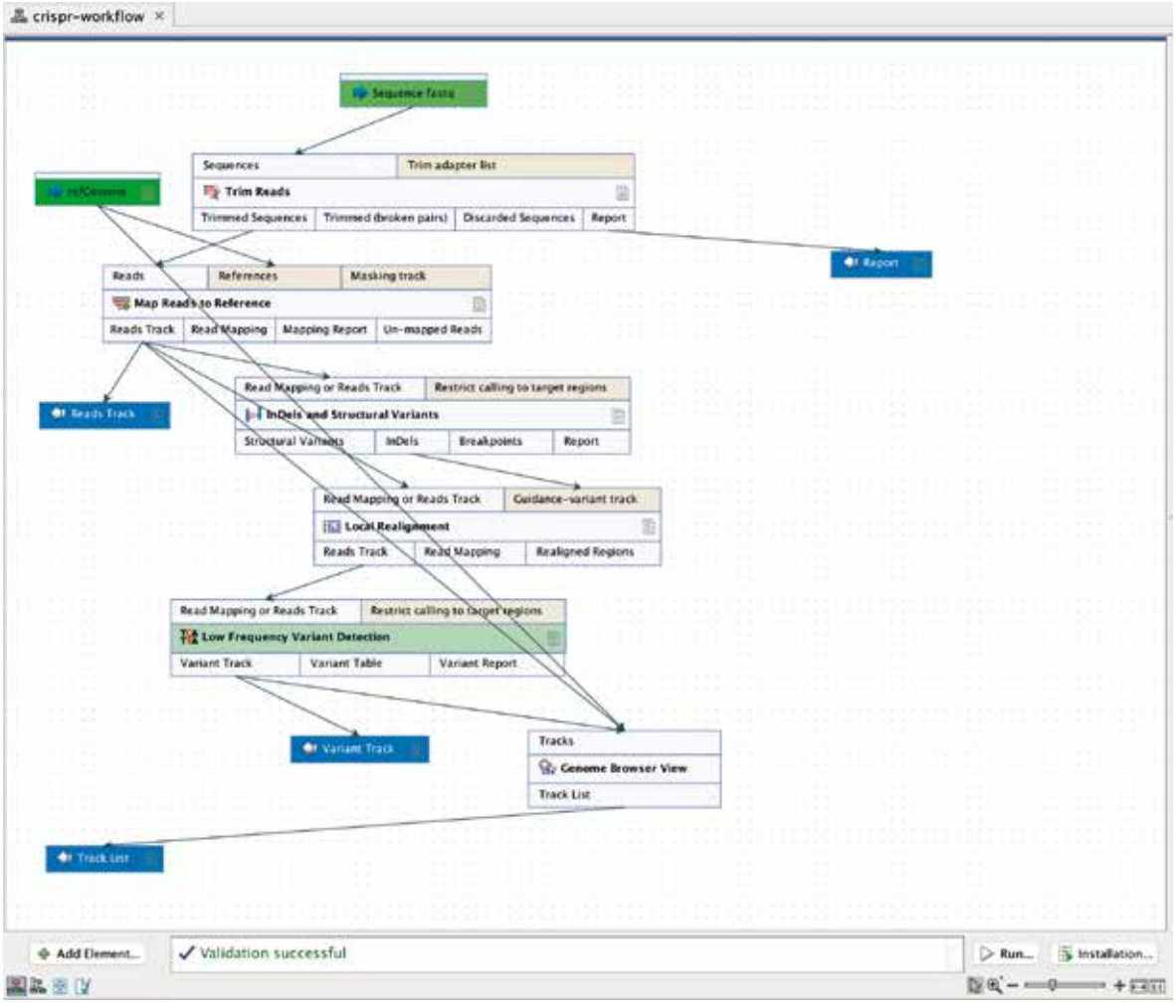

Figure 5. Example of workflow used for differentially filtering variants, and compared with validated variants (truth) in order to calculate accuracy (F1 score)

In order to create all possible combinations, and to optimize the filtering parameters, we have created six different arrays containing the values we want to test. We then launched the workflow inside nested loops which created the combinations of all values. An example of the code is shown in Code Box 6.

In the following example, we have used acronyms for each parameter: average quality (AvQ), read direction test (RDT), read position test (RPT); we have appended the parameter acronym to the variant type we used it for: small for SNVs and large for INDELs.

Code Box 6: Optimizing variant filters in Bash, in a set of nested loops

```
Code Box 6: Optimizing variant filters in Bash, in a set of nested loops
AvQSmallArray=( 30 35 40)
RDTSmallArray=( 0.00001 0.0000001 )
RPTSmallArray=( 0.00001 0.0000001 )
AvQLargeArray=( 25 35 )
RPTLargeArray=( 0.01 0.00001 )
RDTLargeArray=( 0.01 0.00001 )

for AvQSmall in "${AvQSmallArray[@]}"
do
for RDTSmall in "${RDTSmallArray[@]}"
do
for RPTSmall in "${RPTSmallArray[@]}"
do
for AvQLarge in "${AvQLargeArray[@]}"
do
for RPTLarge in "${RPTLargeArray[@]}"
do
for RDTLarge in "${RDTLargeArray[@]}"
do
BaseName="AvQSmall${AvQSmall}_RDTSmall${RDTSmall}_RPTSmall${RPTSmall}_AvQLarge${AvQLarge}_
RPTLarge${RPTLarge}_RDTLarge${RDTLarge}"
echo "testing now $BaseName"
echo "testing now $BaseName" >>loop_log.txt
```

```
clcserver \
-S yourserver \
-U youruser \
-W XXX-PASS-TOKEN-AAAA-BBB \
-A wf-company-workflow-name \
-G your_grid_queue \
–filter-input-workflow-input clc://yourserlver:7777/your_location/file/name \
-d clc://yourserlver:7777/your_location/file/var-filtering \
–filtered-variant-track-filter-output "/${BaseName}/${BaseName}_filtered" \
–identify-indels-to-be-removed-indel-criteria "any{Average quality < $AvQLarge}{Read position test probability < $RPTLarge}
{Read direction test probability < $RDTLarge}" \
–identify-snvs-to-be-removed-snv-criteria "any{Read position test probability < $RPTSmall}{Read direction test probability <
$RDTSmall}{Average quality < $AvQSmall}" \
–variant-track-2-fns "/${BaseName}/${BaseName}_FNs" \
–variant-track-fps "/${BaseName}/${BaseName}_FPs" \
–variant-track-1-tps "/${BaseName}/${BaseName}_TPs"
done
done
done
done
done
done
```

Executing the same workflow inside six nested loops generates a large number of combinations of parameters: for each combination we counted the number of TPs, FPs and FNs, calculated sensitivity and precision, and finally computed the accuracy (F1 score). This approach allows one to select the best combination of filtering parameters to optimise the accuracy of variant calling and filtering.

In addition to selecting the best combination of parameters, the generation of a large set of results can also be used to further investigate the importance of each of the parameters on the accuracy of the results. To this aim, the data was analysed using R with the Boruta package[2], which has developed an algorithm extending the Random Forest approach. This method creates "shadow variables" for each attribute by random shuffling its values across the objects, and using these artificial variables to improve the estimation of the importance of each parameter.

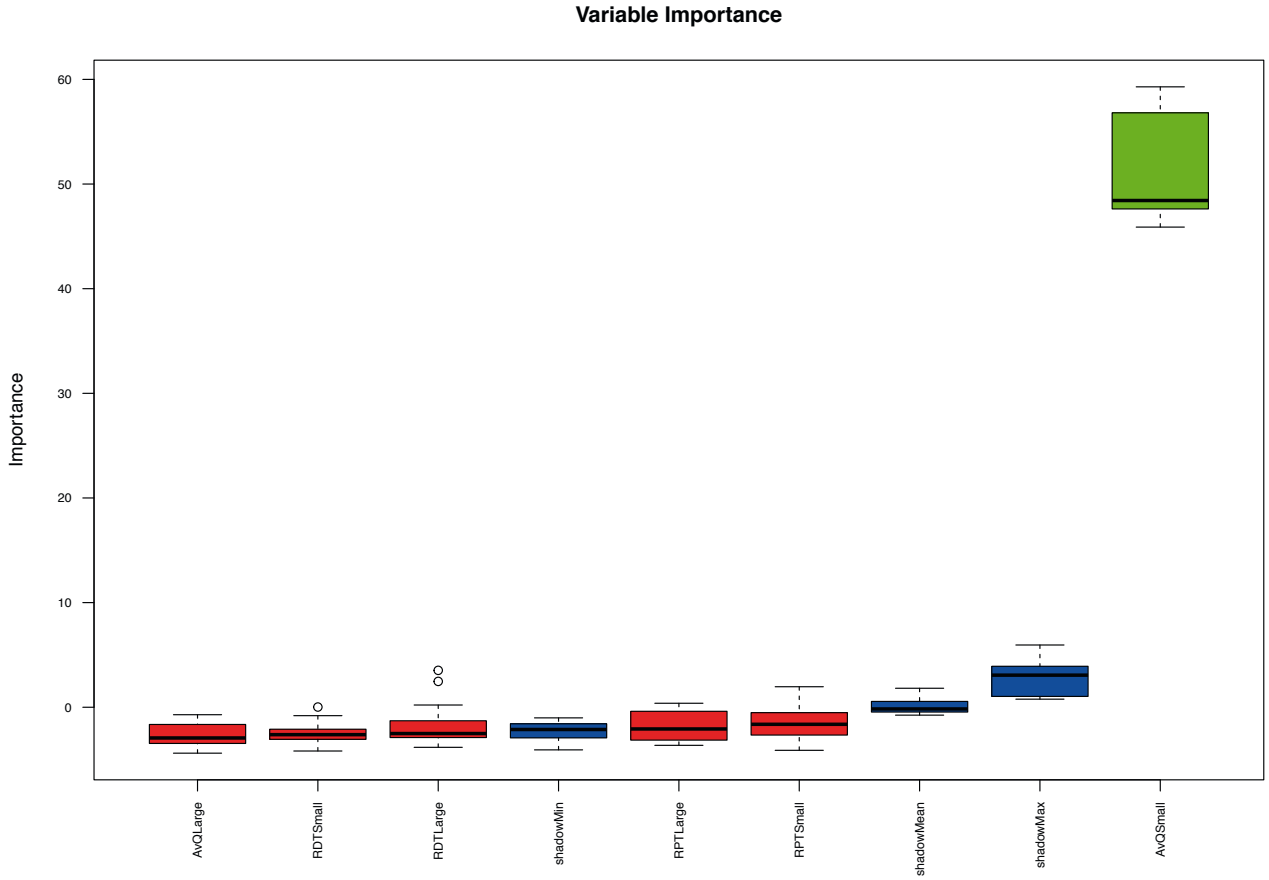

Figure 6. Output of Boruta package in R, showing the importance of the different parameters in influencing the accuracy of the results

In our hypothetical example, we have identified the threshold for the average quality, applied to SNVs, as the parameter that most influences the accuracy of the results.

Conclusions

In this application note we demonstrate the flexibility and power of the CLC Server Command Line Tools. Users with a minimum knowledge of the command line, or any programming language, can quickly access all features available in the CLC Genomics Server, without the use of a graphical interface.

CLC Server Command Line Tools represent an opportunity to address the needs of a more demanding bioinformatics environment and enterprise scenarios, where automation, integration with other pipelines and optimization of the workflows are particularly important.

The examples in this application note can be used as a template for further developing the code, thus allowing users to quickly get started and get the most out of this additional tool.

References

1. Güell, M., Yang, L. & Church, G. M. Genome editing assessment using CRISPR Genome Analyzer (CRISPR-GA). Bioinformatics 30, 2968–2970 (2014).
2. Kursa, M. B. & Rudnicki, W. R. Feature Selection with the Boruta Package. Journal of Statistical Software 36, 1–13 (2010).

Discover more at **www.qiagenbioinformatics.com**

To learn more, have a look at these informative tools:

Web resources

CLC Genomics Workbench

www.qiagenbioinformatics.com/products/clc-genomics-workbench

CLC Server Command Line Tools

www.qiagenbioinformatics.com/products/clc-server-command -line-tools

CLC Microbial Genomics Module

www.qiagenbioinformatics.com/products/clc-microbial-genomics-module

Tutorials

www.qiagenbioinformatics.com/support/tutorials

For up-to-date licensing information and product-specific disclaimers, see the respective QIAGEN kit handbook or user manual. QIAGEN kit handbooks and user manuals are available at www.qiagen.com or can be requested from QIAGEN Technical Services or your local distributor.