CLC Server Command Line Tools

User manual

Manual for

CLC Server Command Line Tools 1.2

Windows, Mac OS X and Linux

May 15, 2011

This software is for research purposes only.

CLC bio
Finlandsgade 10-12
DK-8200 Aarhus N
Denmark

Contents

Chapter 1

Introduction

The *CLC Server Command Line Tools* is a command-line client for the CLC Genomics Server. It is able to start analyses on the server, import and export data, and perform various data operations such as moving, renaming and deleting data on the server. A typical work flow using the *CLC Server Command Line Tools* might be:

1. Import your sequence data
2. Run analyses such as read mapping, SNP detection or RNA-Seq
3. (Optionally) export the results to you local disk

Another client available to run tasks on the CLC Genomics Server is the graphical *CLC Genomics Workbench*. Below are recommendations for choosing which of these two clients, the graphical or the command line, to use for your work:

- For *visualization and interpretation of data* we recommend the *CLC Genomics Workbench*. The only way to visualize and interpret data when you have worked using the *CLC Server Command Line Tools* is to export the results into file formats that can be imported into visualization tools.
- For *explorative work* we recommend using the *CLC Genomics Workbench*. The numerous parameters are easier to interpret using the graphical interface, and selection and management of data is more intuitive through this interface for most users. In addition, the graphical user interface has more constraints to help guide reasonable choices of parameters and combination of parameters; these constraints are not all present in the *CLC Server Command Line Tools*.
- For automation and consistency, of particular utility in production environments, the *CLC Server Command Line Tools* client is recommended. In particular, you can *script pipelines* of analyses on the CLC Genomics Server, and then use these scripts for processing many data sets in a consistent manner. For initial pilot runs, it is often helpful to use the exploratory features of the *CLC Genomics Workbench* to determine quality control and parameter settings, and then incorporate these settings into a script using the *CLC Server Command Line Tools*.

This user manual begins with installation instructions followed by an explanation of the basics of operating the *CLC Server Command Line Tools*. Then, we provide an example script, which illustrates various aspects of how to use the analysis tools available on the server.

1.1 Installation

The *CLC Server Command Line Tools* can be downloaded from http://www.clcbio.com/download_clt_beta and is available for Windows, Mac and Linux. You can install the tools on any computer that can connect to the your CLC Genomics Server, but it makes sense to install them onto the computer that will be used to run the scripts, or onto the server computer itself.

1.2 System requirements

The system requirements of *CLC Server Command Line Tools* are these:

- Windows 2000, Windows XP, Windows Vista or Windows 7
- Mac OS X 10.4 or newer
- Linux: Redhat or SuSE
- 32 or 64 bit
- 256 MB RAM required
- 512 MB RAM recommended
- 1024 x 768 display recommended

You will also need a running version of *CLC Genomics Server*. No additional license is required for running the *CLC Server Command Line Tools*.

Chapter 2

Basic usage

Once installed, there will be three programs present in the installation folder:

- `clcserver` - the key program. It is used to run all the commands that communicate with the server.
- `clcresultparser` - used to parse data locations from particular text files generated during clcserver runs. This command is most useful when connecting analyses in a scripting pipeline (see section 2.3).
- `clcserverkeystore` - a helper tool for enabling passwords to be handled securely (see section 2.1).

The `clcserver` program requires the following four flags, which provide information about the connection to the server:

–S <hostname or IP address of the server>

–P <port the server runs on> When omitted, port 7777 is used, which is the default for server installations.

–U <user name> The username to log into the server with.

–W <password or token> See section 2.1 for how to avoid entering passwords in clear text.

The commands to be run on the server are supplied with the flag:

–A <command to be executed on server>

If you supply the -A flag with no parameter, a list of all commands that can be run on the server will be returned.

If you supply the -A flag with a program name, but do not provide the required flags for that program, then a listing of the flags for that program will be returned. For example, a command of a form like:

```
clcserver –S server.com –U bob –W secret –A read_mapping
```

would return the full list of parameters for the `read_mapping` function, including the possible values, and descriptions. This information, for each command, is also available in the online manual at http://www.clcbio.com/index.php?id=1641&manual=Usage_all_commands.html).

An optional flag when working on the command line, but important when working with scripts, is:

–O <filename> The name of a file to be created to hold a summary of steps carried out on the server and data locations of the results generated. The data locations are of a form that can be used by downstream CLC commands. See section 2.3 for information about parsing this file. By default, this file is placed in your working directory. If you do not provide this flag, this data will be written to a file called `results.txt`

For those working with the *CLC Grid Integration Tool (Beta)*, you can run import and algorithm commands through your grid nodes by adding the following flag to your clcserver command:

–G <grid preset name>

Please note that export commands cannot be run using your grid nodes at this time.

Other optional flags available for the clcserver command are:

–C <integer> Specify the **column width** of the help output.

–D <boolean> Enables **debug** mode when set to true, providing more elaborate output and error messages.

–H Display general **help** instructions.

–V Display the **version number** of *CLC Server Command Line Tools*.

2.1 Handling passwords

To help you avoid sending your server login password in clear text across the network, we provide the `clcserverkeystore` tool. This enables you to convert your password to a token, which is stored and can be interpreted by the *CLC Server Command Line Tools* when logging in to the server. The token is encrypted and saved with the user profile on the computer running the *CLC Server Command Line Tools*.

You can generate a password token using the following command:

```
clcserverkeystore --generate
```

You will be prompted for the password. After you type the password in, press the **Enter** key. The password token is then returned on screen. It will be quite a long string of text, that you should save somewhere to refer to for future use.

So, if we say that user `bob` has password `secret`, and has generated a password token `CAIHMAAAAAAAAAPcb769377f4`, then he could enter either of the following two commands to connect to his server. The first passes the password in plain text. The second, passes it as an encrypted token.

```
clcserver -S server.com -U bob -W secret
```

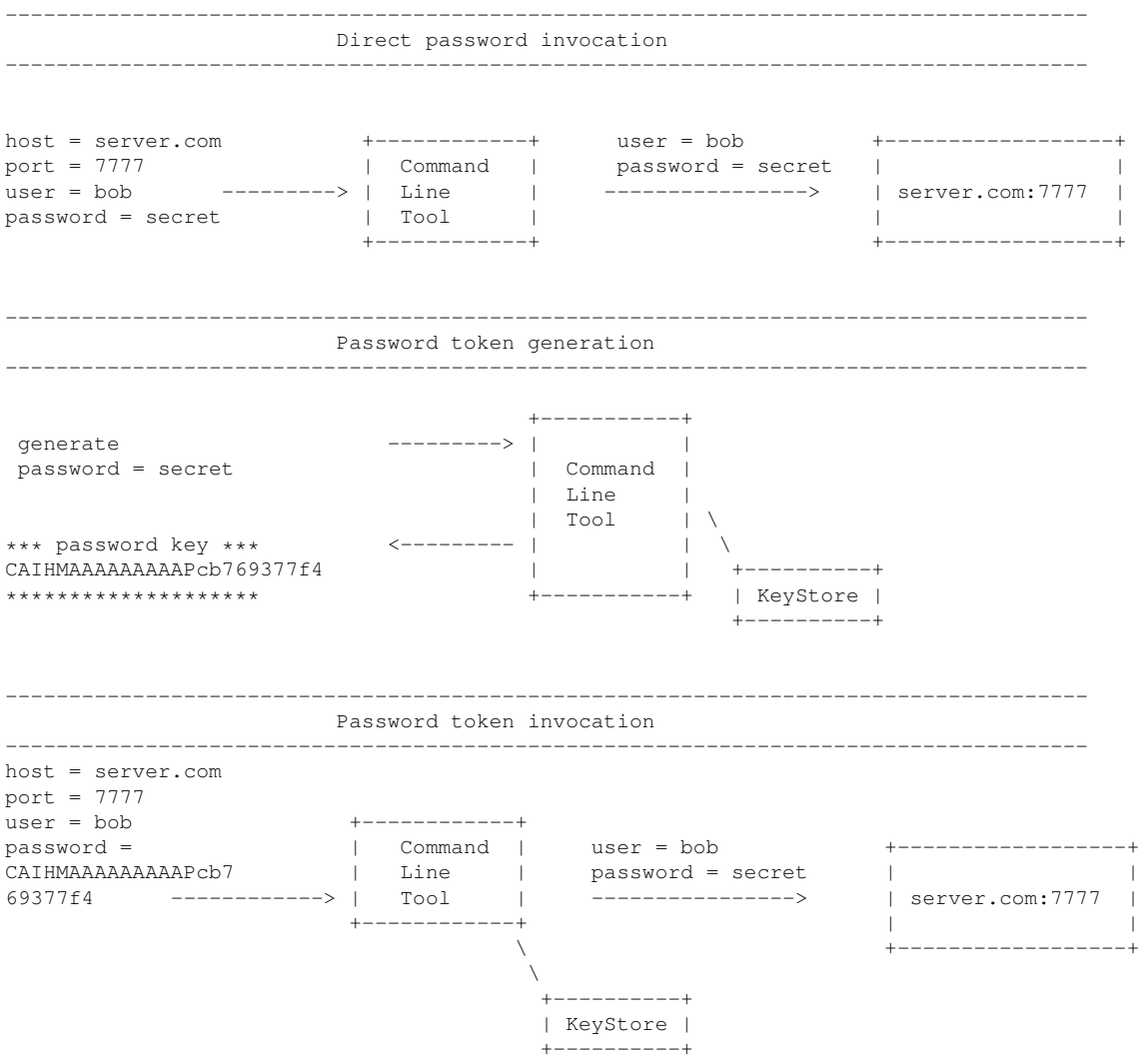
```
clcserver -S server.com -U bob -W CAIHMAAAAAAAAAPcb769377f4
```

If the token needs to be deleted, the `clcserverkeystore` program has two other parameters that can be used:

-d <token> This will delete the individual token provided as a parameter.

deleteAll This will delete all the tokens in the user profile.

The first section of the diagram below illustrates the process of logging into the server using a clear text password. The second section illustrates the process of generating a password token and storing it in the keystore, followed by a section showing how the token is substituted by the *CLC Server Command Line Tools* with the real password when initiating the connection to the server.



2.2 Referring to files: the CLC URL

Most commands will refer to data stored in files. For some tasks such as importing or exporting, the files may be on your local computer system. For other tasks, like running analyese, the data may already be uploaded to the CLC server.

2.2.1 Data on your local system

Certain commands, in particular import and export commands, will require you to provide files on your local system. Here, you need only provide the *relative or full path* to the file or directory of interest on your system.

The file named as the parameter for the -O option of the `clcserver` command is another example where you provide a location on your local file system.

2.2.2 Data on the server

For files already on machine housing the CLC Genomics Server, you need to provide the data location as a CLC URL. These start with the text `clc://`.

Files already uploaded to the CLC Genomics Server data area:

For files already available within the CLC Genomics Server, you have a choice of providing a CLC URL with the (human-readable) path to your file, or providing a CLC URL with an ID that the CLC software understands. The former is easier when working on the command line directly. The latter is used primarily when working with pipelines of operations, for example when scripting.

IDs of data locations look something like this: `BAAAAAAPc132--7fff/-268177574-7fff`. The first part of this example ID gives the data location (`BAAAAAAPc132--7fff`), and this is followed by the ID for the file (`-268177574-7fff`). The advantage of using IDs over the path to the file is that the ID of the data remains the same, even the data object is renamed or moved. If you wish to work with data IDs on the command line, you can be easily get these using the Workbench. See figure 2.1.

Examples of providing file locations for data uploaded to the CLC Genomics Server data area include:

- `clc://server/server_data/project1/sample1`. This refers to a file called `sample1` on the `server`, located in the `server_data` data location under the `project1` folder.
- `clc://server/BAAAAAAPc132--7fff/-268177574-7fff`. This URL is also pointing to a file on the `server`, using an ID to refer to it.
- `clc://server.com:7777/server_data/project1/sample1`. This is an example of using the server host name and port rather than just referring to `server`. Since you need to be logged in to a CLC Genomics Server to get access to the data, this form is generally not necessary.

Files ready for upload to the CLC Genomics Server:

You can also use data files on the same system as the CLC Server, where that data has not been imported into a designated server data area. Data you plan to use this way, that is, use

without explicitly importing it, needs to be in a location that the Server can access. This means it must be within a designated **High-throughput sequencing data import location**.

To use data in one of these areas, the `serverfile` version of a CLC URL is provided: `clc://serverfile`. For example:

- `clc://serverfile/mnt/data/project1/s_1_1.sequence.txt`. This refers to a file located on the same machine as the CLC Genomics Server; the full path to the file is /mnt/data/project1/s_1_1.sequence.txt .

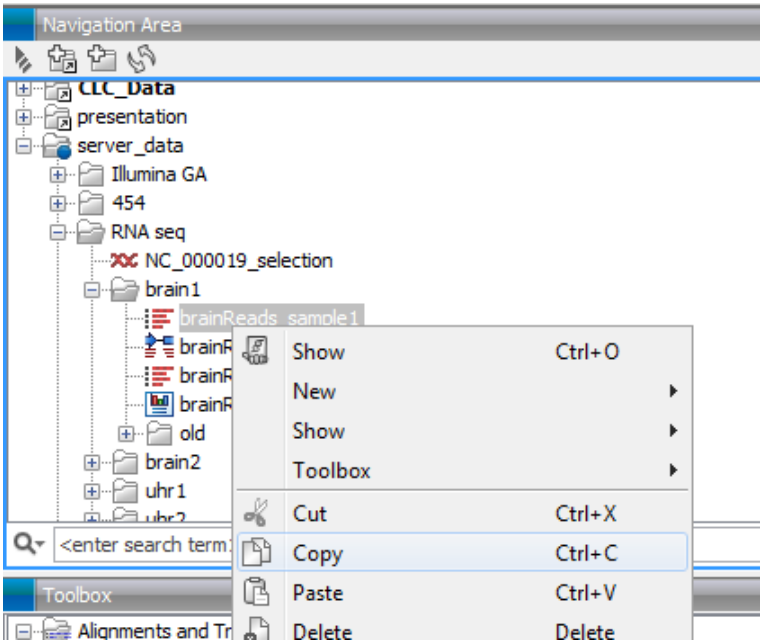


Figure 2.1: *Copying a data object in the workbench will put the CLC URL on the clipboard. You can then paste the URL into your command in the terminal.*

2.3 Result files and connecting analyses in pipelines

For each run of `clcserver`, text information is returned providing a summary of the steps taken, and the locations, in ID form, of any files generated. The file containing this information will, by default, be created in the *current directory* and will be called `result.txt`. You can use the `-O` option for the `clcserver` command if you wish to specify an alternative file to be written to.

An example of the type of contents you will see in a typical result file is shown below. In this case it is a file that was generated after running the the trim algorithm using a sequence list called `reads` as input. The result file lists the three files that were produced.

```
//
Name: reads trimmed
ClcUrl: clc://127.0.0.1:7777/-268177574-YCAAAAAAAAAAAAPc673b0db8c5e724f--5d66a991-12d75090d93--7fff
//
//
Name: reads report
ClcUrl: clc://127.0.0.1:7777/-268177574-ADAAAAAAAAAAAAPc673b0db8c5e724f--5d66a991-12d75090d93--7fff
//
//
Name: Trim Sequences log
ClcUrl: clc://127.0.0.1:7777/-268177574-CAAAAAAAAAAAAAPc673b0db8c5e724f--5d66a991-12d75090d93--7fff
//
```

When creating pipelines stitching together several analyses, you parse the result file to get the location of the data produced, which is needed as input for the next algorithm.

The result file is just a text file, but it can still be a challenge to parse it to get the necessary CLC URLs. Thus, we provide a tool called `clc_result_parser` to help with this. It searches the result file for a text expression you provide, and returns the CLC URL for files where a match to that text has been found in the `Name:` field. The `Name` field will using contain the name of the input data along with a description of the type of data held in that file location.

In the case above, you would probably search for the trimmed reads to use for further analysis, which could be done with a command like this:

```
clcresultparser -f result.txt -c trimmed
```

Here, the following text would be returned:

```
clc://127.0.0.1:7777/-268177574-YCAAAAAAAAAAAAPc673b0db8c5e724f--5d66a991-12d75090d93--7fff
```

The options for the `clcresultparser` program are:

–f <name of result file to parse> This option is required.

–c <text to search for> Text to search for in the `Name` field of the result file. If nothing is found, the exit code is 1.

–n <text that should not match> Text that should **not be contained** in the `Name` field of the result file.

–r <regexp> A **Java regular expression** used for matching the name of the output (see `http://java.sun.com/docs/books/tutorial/essential/regex/index.html`).

– –ignorelogs <boolean> By default, all analyses produce log files. You can provide `false` as the argument to this option to stop log files from being returned. This is equivalent to excluding all names ending with `log`, or `log` with a number suffix. The latter are generated when there is more than one log file in the same folder.

–p <prefix text> When more than one match is found, the data locations for all matches will be output as a space-separated list. By supplying a **prefix** string, you can stipulate what character(s) to separate the list using. E.g. If you need to send several files output by the `clcresultparser` command as arguments to `-i` options for the next analysis, simply provide `"-i"`as as the argument for the `-p` flag.

–e <integer> The number of CLC URLs that are **expected** to be returned. If this is not the number of results files that match the search string. the command will return with exit code 10. This option is designed for use in scripts where you will wish to carry out validation steps are you proceed through the pipeline. (On the command line, you check the error code returned by the previous command by typing `echo $?`.

–C <integer> Specifies the **column width** of the help output.

Chapter 3

Example script

In this section, we present an example script showing a typical work flow consisting of the following steps:

- Import of NGS data
- Read mapping
- SNP detection
- DIP detection

The result of the SNP and DIP detection are then exported to the local file system in Excel format.

The script is intended only as an example. Hopefully it will be possible for you to modify it to fit your purposes.

You can download the script including data from
http://download.clcbio.com/CLCServerCommandLineToolsBeta/1.0/example-workflow.zip.

This is a shell script that will run on Linux and Mac OS.

```
#!/bin/sh
##############################################################################################
## Example workflow script for CLC Server Command Line Tools 1.0 beta 1
## CLC bio, January 2011
##
## For full documentation please visit: http://clcbio.com/usermanuals
##############################################################################################


### SETTINGS (Edit before use) ###############################################################

# 1. Configure your server connection parameters

SERVER="node01"
PORT="7777"
USER="root"
PASSWORD="default"


# 2. Configure the path to the CLC Server Command Line tools

SERVERCMDPATH="/Applications/clcservercmdline/clcserver"
```

```
PARSECMDPATH="/Applications/clcservercmdline/clcresultparser"


# 3. Configure "High-throughput sequencing data import" location for import data
#  - Use the server web-interface to setup a data import location
#  - Copy the example files from the data directory to this location
#  - Edit the IMPORTPATH variable below to a CLC URL point to this location
#
# For more info please visit:
# http://www.clcbio.com/index.php?id=1484&manual=Direct_import_high_throughput_sequencing_data.html
#
# For information about CLC URLs please visit:
# http://www.clcbio.com/index.php?id=1641&manual=Referring_files_CLC_URL.html

IMPORTPATH="clc://serverfile/mnt/data/temporary_data/cmdline"


# 4. Configure data paths for saving results
#  - Use the server web-interface to configure a file system location for data storage
#  - Edit the DATAPATH variable below to a CLC URL pointing to this location
#
# For more info please visit:
# http://www.clcbio.com/index.php?id=1484&manual=Adding_file_system_location.html
#
# For information about CLC URLs please visit:
# http://www.clcbio.com/index.php?id=1641&manual=Referring_files_CLC_URL.html

DATAPATH="clc://server/test"
DIR="clc-example"
SUBDIR="workflow-example"



### FUNCTIONS ###################################################################################

function check_return_code {
return_code=$?
cmdname=$1
#echo Return code: $return_code
if [ $return_code -ne 0 ]
then
echo ""
echo "### Error during: $cmdname ###"
echo "Terminating script"
exit 1
fi
}

### COMMANDS ####################################################################################
SERVERCMD="$SERVERCMDPATH -S $SERVER -P $PORT -U $USER -W $PASSWORD"
PARSECMD=$PARSECMDPATH


### WORKFLOW SCRIPT #############################################################################

# Make a directory for DIR
echo ""
echo "Make a directory: $DIR"
$SERVERCMD -A mkdir -t ${DATAPATH} -n ${DIR} -O tmpdir_result.txt
check_return_code "make dir"
tmpdir=`$PARSECMD -f "tmpdir_result.txt" -c "clc-example"`
check_return_code "make dir result parsing"
rm tmpdir_result.txt


# Make subdirectory in DIR folder for result and data
echo ""
echo "Make subdirectory: $SUBDIR"
$SERVERCMD -A mkdir -t ${tmpdir} -n ${SUBDIR} -O mkdir_result.txt
```

```
check_return_code "Make sub dir"
destdir=`$PARSECMD -f mkdir_result.txt -p "-d" -c $SUBDIR`
check_return_code "Make sub dir result parsing"
rm mkdir_result.txt


# Import solid data
echo ""
echo "Import solid data"
$SERVERCMD -A ngs_import_solid -f "$IMPORTPATH/solid_matepair_F3.csfasta" \\
-f "$IMPORTPATH/solid_matepair_F3._QV.qual" -f "$IMPORTPATH/solid_matepair_R3.csfasta"\\
 -f "$IMPORTPATH/solid_matepair_R3._QV.qual" --paired-reads true\\
  --read-orientation FORWARD_FORWARD ${destdir} -O ngs_import_solid_result.txt
check_return_code "Import solid data"
soliddata=`$PARSECMD -f ngs_import_solid_result.txt -p "-i" -c "solid" --ignorelog true`
check_return_code "Import solid data result parsing"
rm ngs_import_solid_result.txt

# Import genome
echo ""
echo "Import genome"
$SERVERCMD -A import -f automaticimport -s "$IMPORTPATH/reference.fa" ${destdir} -O import_result.txt
check_return_code "Import genome"
refdata=`$PARSECMD -f import_result.txt -p "--references" -c "reference"`
check_return_code "Import genome result parsing"
rm import_result.txt


# Do readmapping
echo ""
echo "Read Mapping"
$SERVERCMD -A read_mapping --maximum-distance 25000 ${soliddata} ${destdir} ${refdata}\\
 -O read_mapping_result.txt
check_return_code "Read Mapping"
readmap=`$PARSECMD -f read_mapping_result.txt -p "-i" -c "mapping"`
check_return_code "Read Mapping result parsing"
rm read_mapping_result.txt


# DIP detection
echo ""
echo "DIP detection"
$SERVERCMD -A dip_detection --create-table true ${readmap} ${destdir} -O dip_result.txt
check_return_code "DIP Detection"
diptable=`$PARSECMD -f dip_result.txt -p "-s" -c "Table"`
check_return_code "DIP Detection result parsing"
rm dip_result.txt


# Export dip table to excel
echo ""
echo "Export DIP table to excel"
$SERVERCMD -A export -e excel_2010 -d . ${diptable} -O dip_export_result.txt
check_return_code "Export DIP table to excel"
dipfile=`$PARSECMD -f dip_export_result.txt`
rm dip_export_result.txt


# SNP detection
echo ""
echo "SNP detection"
$SERVERCMD -A snp_detection ${readmap} ${destdir} -O snp_result.txt
check_return_code "SNP Detection"
snptable=`$PARSECMD -f snp_result.txt -p "-s" -c "Table"`
check_return_code "SNP Detection result parsing"
rm snp_result.txt


# Export snp table to excel
```

```
echo ""
echo "Export SNP table to excel"
$SERVERCMD -A export -e excel_2010 -d . ${snptable} -O snp_export_result.txt
check_return_code "Export SNP table to excel"
snpfile=`$PARSECMD -f snp_export_result.txt`
rm snp_export_result.txt

# Workflow completed
echo "Workflow completed succesfully"
echo "dip_detection result: ${dipfile}"
echo "snp_detection result: ${snpfile}"
```