culebrONT

Documentation

Release 1.3.2

Julie Orjuela (IRD),
Aurore Comte (IRD),
Sebastien Ravel (CIRAD),
Florian Charriat (CIRAD)

Feb 15, 2021

Getting started

How to install CulebrONT ?

CulebrONT uses

In your local computer OR on HPC cluster, first of all clone our repository or download last version from https://github.com/SouthGreenPlatform/CulebrONT_pipeline

```
git clone https://github.com/SouthGreenPlatform/CulebrONT_pipeline.git
cd CulebrONT_pipeline
```

In order to build a pipeline, you have to adapt two files in YAML (Yet Another Markup Language) format `tools_path.yaml` and then `config.yaml`. The snakemake pipeline will then execute the set of rules executed into their own virtual environment.

For installation, you need to first modify `tools_path.yaml` file. You need to adapt path to singularity images and also conda environments. CulebrONT integrates many tools. In order to build a workflow, CulebrONT needs to know where tools are installed. You can give only tools you will use for your analysis. To avoid you to install them, we provide singularity images and conda environments. We will explain all here!

How to build singularity images?

You have three options to obtain singularity build images. Choose one of them!

1. Give path to **singularity hub** images from our CulebrONT repository. If you use singularity hub repository build singularity images will be download from the cloud :

```
SINGULARITY:
    REPORT : 'shub://SouthGreenPlatform/CulebrONT_pipeline:report.def'
    SHASTA : 'shub://SouthGreenPlatform/CulebrONT_pipeline:shasta-0.7.0.def
↪'
    WEESAM : 'shub://SouthGreenPlatform/CulebrONT_pipeline:weesam-latest.
↪def'
    ASSEMBLYTICS : 'shub://SouthGreenPlatform/CulebrONT_
↪pipeline:assemblytics-1.2.def'
    MEDAKA : 'shub://SouthGreenPlatform/CulebrONT_pipeline:medaka-gpu-1.2.
↪def'
    BLOBTOOLS : 'shub://SouthGreenPlatform/CulebrONT_pipeline:blobtools-1.
↪1.1.def'
    KAT: 'shub://SouthGreenPlatform/CulebrONT_pipeline:kat-latest.sif'
    BUSCO: 'shub://SouthGreenPlatform/CulebrONT_pipeline:busco-4.1.4.sif'
```

2. You can build available *.def* recipes available on the *CulebrONT_pipeline/Containers* repertory. Feel free to build them on your own computer (or cluster); be careful, you need root rights to do it. Use :

```
cd Containers/
sudo make build
```

Now give to CulebrONT path from build images on `tools_path.yaml` such as :

```
SINGULARITY:
    REPORT : './Containers/Singularity.report.sif'
    SHASTA : './Containers/Singularity.shasta-0.7.0.sif'
    WEESAM : './Containers/Singularity.weesam-latest.sif'
    ASSEMBLYTICS : './Containers/Singularity.assemblytics-1.2.sif'
    MEDAKA : './Containers/Singularity.medaka-gpu-1.2.sif'
    KAT : './Containers/Singularity.kat-latest.sif'
    BLOBTOOLS: './Containers/Singularity.blobtools-1.1.1.sif'
    BUSCO: './Containers/Singularity.busco-4.1.4.sif'
```

3. Download build available singularity images from i-Trop server https://itrop.ird.fr/culebront_utilities/. Use

```
cd CulebrONT_pipeline/Containers
wget -rm -nH --cut-dirs=2 --reject="index.html*" --no-parent https://itrop.
↪ird.fr/culebront_utilities/singularity_build/
```

Note: If you don't want to use some tool, please leave the path empty. Like bellow for SHASTA and ASSEMBLYTICS and don't remove the line of the tool PLEASE.

```
SINGULARITY:
    REPORT : 'path/to/Containers/Singularity.report.sif'
    SHASTA : ''
    WEESAM : 'path/to/Containers/tools/Singularity.weesam-latest.sif'
    ASSEMBLYTICS : ''
    MEDAKA : 'path/to/Containers/Singularity.medaka-gpu-1.2.sif'
    BLOBTOOLS : 'path/to/Containers/Singularity.bloobtools-1.1.1.sif'
    KAT: 'path/to/Containers/tools/Singularity.kat-latest.sif'
```

How to build use Conda environments?

A series of Conda environment are available on our repository for each tool used. These environments will be automatically build the first time that you launch CulebrONT. Conda are available on the *CulebrONT_pipeline/envs/* folder. Give to CulebrONT conda path required on *tools_path.yaml* file such as :

```
CONDA:
    FLYE : './envs/flye-2.8.1.yaml'
    CANU : './envs/canu-2.0.yaml'
    MINIPOLISH : './envs/minipolish-0.1.2.yaml'
    RAVEN : './envs/raven-1.2.2.yaml'
    SMARTDENOVO : './envs/smartdenovo-1.0.0.yaml'
    CIRCLATOR : './envs/circlator-1.5.2.yaml'
    R : './envs/R_for_culebront.yaml'
    DIAMOND : './envs/diamond.2.0.4.yaml'
    MUMMER : './envs/mummer-3.23.yaml'
    MAUVE : './envs/mauve-2.4.0.yaml'
    MINIASM_MINIMAP2 : './envs/miniasm-0.3_minimap2-2.17.yaml'
    MINIMAP2_SAMTOOLS : './envs/minimap2-2.17_samtools-1.10.yaml'
```

(continues on next page)

(continued from previous page)

```
    RACON_MINIMAP2 : './envs/racon-1.4.13_minimap2-2.17.yaml'
    QUAST: './envs/quast-5.0.2.yaml'
    NANOPOLISH_MINIMAP2_SAMTOOLS_SEQTK : './envs/nanopolish-0.13.2_
↪minimap2-2.17_samtools-1.10_seqtk-1.3.yaml'
```

Danger: conda enviroments are compiled by Snakemake in each output analysis folder. To avoid this, please use `--conda-prefix /path/to/build_conda_env` on snakemake command line.

HPC specifications

To install CulebrONT on a global way, we recommend to charge required dependencies from CulebrONT and modifying *cluster_config.yaml* file.

Preparing *cluster_config.yaml*

On `cluster_config.yaml` , you can add partition, memory and threads to be used by default for each rule. If more memory or threads are requested, please adapt the content of this file before running on a cluster for every rule.

Warning: please adapt the content of this file before running on a cluster for every rule !!

Here is a example of the configuration file we used on the i-Trop HPC.

```
__default__:
    cpus-per-task : 4
    ntasks : 1
    mem-per-cpu : '2'
    partition : "normal"
    output : 'logs/stdout/{rule}/{wildcards}'
    error : 'logs/error/{rule}/{wildcards}'

run_nanopolish :
    cpus-per-task : 12
    mem-per-cpu : '4'
    partition : "long"

run_canu:
    cpus-per-task : 8
    mem-per-cpu : '8'
    partition : "long"
```

Available data test

Optionally, in order to test install of CulebrONT pipeline, a data test `Data-Xoo-sub/` is available on https://itrop.ird.fr/culebront_utilities/. Feel free to download it using `wget` and put it on CulebrONT repertory.

```
cd CulebrONT_pipeline
wget -rm -nH --cut-dirs=1 --reject="index.html*" --no-parent https://itrop.
↪ird.fr/culebront_utilities/Data-Xoo-sub/
```

Now, it is time to prepare configuration file `config.yaml` file to say to CulebrONT what kind of pipeline you want to create and use it with your data !

Why use CulebrONT ?

Assembly, circularisation, polishing and correction steps are included in CulebrONT, and can be activated (or not) according to user's requests. The most commonly used tools in the community for each step are integrated, as well as various quality control tools. CulebrONT also generates a report compiling information obtained at each step.

From assembly to correction

CulebrONT is really flexible to assembly and circularise (or not) assembled molecules as well as polish and correct assemblies. You can give directives on *config.yaml* file to CulebrONT to generate a modular pipeline.

> **Warning:** Logically, you must launch at least one of assemblers included in CulebrONT and pipe assembly with circularisation, polishing and correction steps as well as with the quality control pipeline.

Assembly

CulebrONT includes (at the moment) six recent and community-validated assemblers.

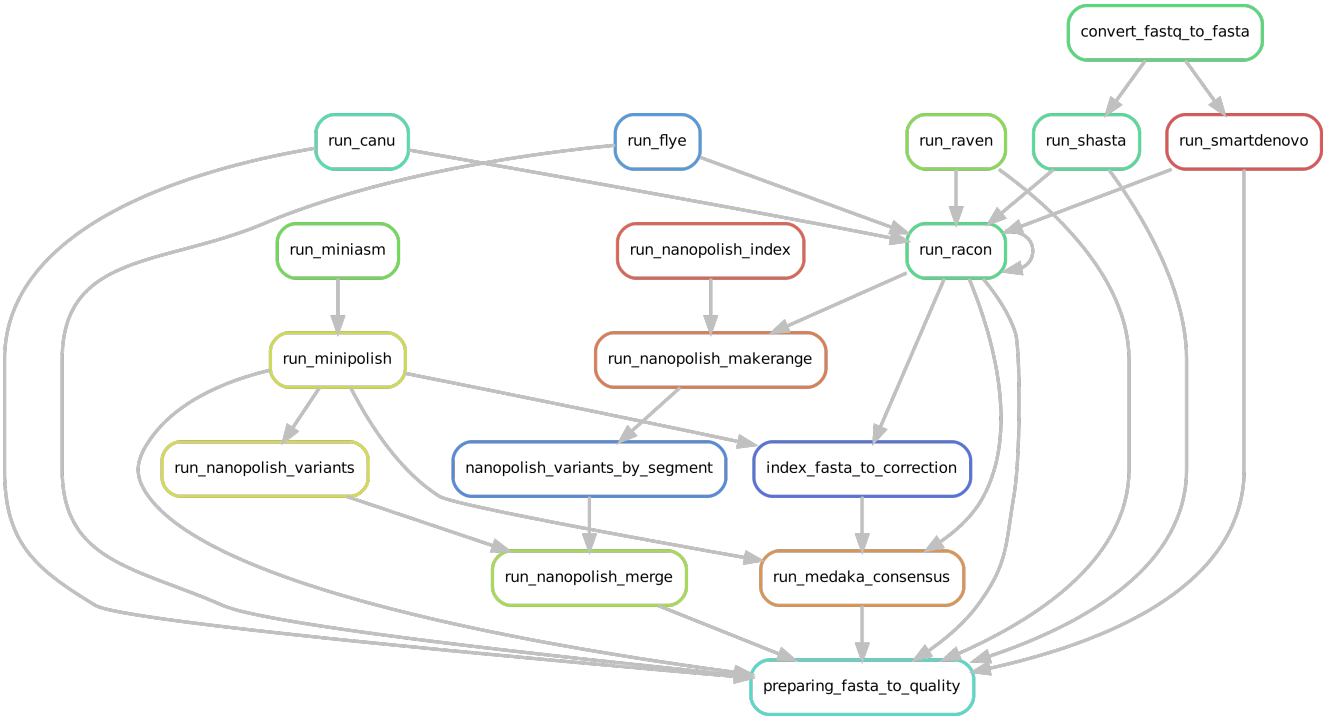
Included tools :

- Flye version >= 2.6 https://github.com/fenderglass/Flye
- Canu version >= 2.0 https://canu.readthedocs.io/en/latest/quick-start.html
- Miniasm version >= 0.3 https://github.com/lh3/miniasm + Minipolish version >= 0.1.2 https://github.com/rrwick/Minipolish
- Shasta version 0.5.1 https://github.com/chanzuckerberg/shasta
- Smartdenovo version 1.0.0 https://github.com/ruanjue/smartdenovo
- Raven version >= 1.2.2 https://github.com/lbcb-sci/raven

Optional circularisation

Using CulebrONT you can activate or deactivate circularisation steps. Typically, if you are interested on eukaryotic organims, circularisation is not necessary, use CIRCULAR=False on `config.yaml` file. Option circular on the configuration file is key on the workflow framework.

Directed acyclic graphs (DAGs) show the differences between deactivated (CIRCULAR=False):



and activated CIRCULAR step on configuration file (CIRCULAR=True):

If an assembled molecule is circular, e.g. for bacteria, this molecule is tagged and will be treated specially in pipeline. We implemented tagging and rotation of circular molecule before each polishing step, and we fixing start position on circular genome. This is efficient when multiple genome alignments are envisaged.

Note:

- If circular step is activated, the *–plasmids* option on Flye is used.
- *Ciclator* tool is used to circularise assemblies from Canu and Smartdenovo. Circlator will attempt to identify each circular sequence and output a linearised version from each of them.
- On circularisation step performed by circlator, trimmed corrected fastq files obtained by Canu are used by circlator. Raw fastq files are used directly for others assemblers.
- Circularisation for Miniasm is performed by minipolish.
- Circularisation for Miniasm, Raven and Shasta is checked using generated GFA files on a special tag_circular step, tagging circular fasta sequences.

OPTIONAL FIXING START

Only if the circular step is activated, a fixstart step is performed before the quality control. This step uses circlator tool with the option fixstart to rotate circular sequences. Rotation is done using the start dnaA gene (if found). This is important when multiple alignments are envisaged.

Fixstart is always performed on the last draft sequenced obtained on the pipeline. In others words:

Note:

- Fixstart is launched after the assembly step if only assembly is activated.
- Fixstart is launched after the polishing step if only assembly+polishing are activated.
- Fixstart is launched after correction if assembly+polishing+correction or assembly+correction are activated

Warning: In any case, Fixstart will be deactivated if CIRCULAR is False

Included tools :

- Circlator version >= 1.5.2 https://github.com/sanger-pathogens/circlator

Polishing

Polishing step is ensured by Racon. Racon corrects raw contigs generated by rapid assembly methods with original ONT reads. Choose how many rounds of Racon you want to perform (constrains from 1 to 9 rounds), and CulebrONT will recursively do it for you. Generally 2 to 4 iterations are the best choices.

Note:

- Minipolish includes racon polishing, so if polishing is deactivated for the others assemblers (flye, shasta, raven, smartdenovo and canu) miniasm will polish anyway, please take it into account to comparisons.
- Raven parameter -p (for polishing) is by default 0, racon is performed on CulebrONT to control rotation of circular molecule before every racon step.

Included tools :

- Racon version >= 1.4.13 https://github.com/isovic/racon

Correction

Correction can improve the consensus sequence for a draft genome assembly. We include *Nanopolish* and *Medaka* on correction steps. With CulebrONT you can train a Medaka model and use it directly to obtain a consensus from you favorite organism.

Note:

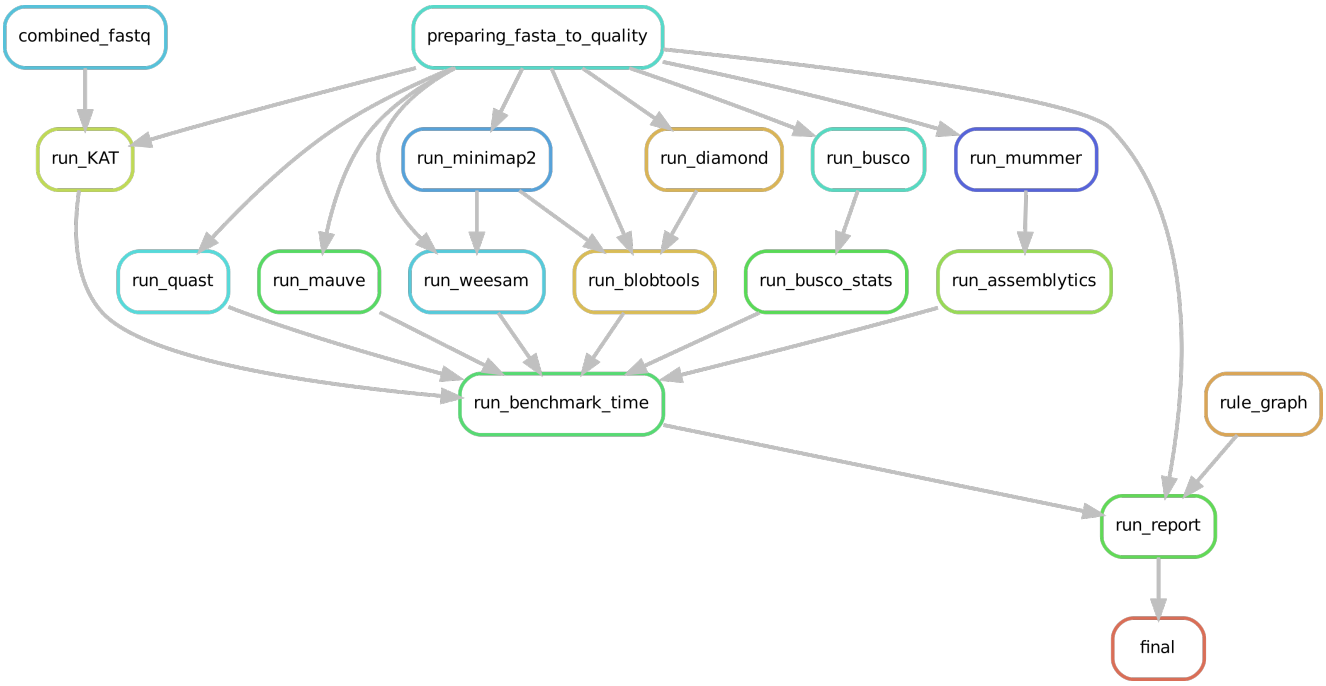
- We have included a split on segments of the assembled molecule before nanopolish and medaka. Each segment is polished on parallel to improve speed and gain time. Segments polished are merged subsequently. CulebrONT has implemented parallelism following medaka documentation and nanopolish practices.

Included tools :

- Medaka Medaka-gpu version >= 1.2 https://github.com/nanoporetech/medaka
- Nanopolish version >= 0.13.2 https://nanopolish.readthedocs.io/en/latest/index.html#

Quality on assemblies

A variety of useful tools exist to check high accuracy assemblies.



CulebrONT checks the quality of the assemblies with using these optional tools:

Note:

- BUSCO : helps to check if you have a good assembly, by searching the expected single-copy lineage-conserved orthologs in any newly-sequenced genome from an appropriate phylogenetic clade.
- QUAST : is a good starting point to help evaluate the quality of assemblies, providing many helpful contiguity statistics.
- Bloobtools : allows to detect contamination on assembled contigs
- Assemblytics to compare contiguity of the assemblies against a reference genome
- KAT to explore kmers frequencies and check possible contaminations

- Weesam can be used to check the coverage of the reads for each assembled contig (for small genome only).
- Multiple alignment of several assembles is performed using Mauve (for small genome only).

Danger: Please, only activate these last two tools for small genomes only.

Included tools :

- BUSCO version >= 4.0.5
- QUAST version >= 5.0.2
- Bloobtools version >= 1.1.1
- Assemblytics version >= 1.2
- KAT version >= 2.4.2
- Weesam version > 1.6
- Mauve > 2.4.0.snapshot_2015_02_13

How to create a workflow ?

CulebrONT allow you to build a workflow using a simple configuration `config.yaml` file. In this file :

- First, provide data paths
- Second, activate tools from assembly to correction.
- Third, activate tools from quality checking of assemblies.
- And last, manage parameters tools.

1. Providing data

First, indicate the data path in the configuration `config.yaml` file:

```
DATA:
    FASTQ: '/path/to/fastq/directory/'
    REF: '/path/to/referencefile.fasta'
    GENOME_SIZE: '1m'
    FAST5: '/path/to/fast5/directory/'
    ILLUMINA: '/path/to/illumina/directory/'
    OUTPUT: '/path/to/output/directory/'
```

Find here a summary table with description of each data need to lauch CulebrONT :

Input	Description
FASTQ	Every FASTQ file should contain the whole set of reads to be assembled. Each fastq file will be assembled independently
REF	Only one REFERENCE genome file will be used by CulebrONT. This REFERENCE will be used for quality steps (ASSEMBLYTICS, QUAST and MAUVE)
GENOME_SIZE	Estimated genome size of the assembly can be done on mega (Mb), giga(Gb) or kilobases (Kb). This size is used on some assemblers (CANU) and also on QUAST quality step
FAST5	Nanopolish needs FAST5 files to training steps. Please give the path of FAST5 folder in the *FAST5* DATA parameter. Inside this directory, a subdirectory with the exact same name as the corresponding FASTQ (before the *.fastq.gz*) is requested. For instance, if in the *FASTQ* directory we have *run1.fastq.gz* and *run2.fastq.gz*, CulebrONT is expecting the *run1/* and *run2/* subdirectories in the FAST5 main directory
ILLUMINA	Indicate the path to the directory with *Illumina* sequence data (in fastq or fastq.gz format) to perform KAT quality. Use preferentially paired-end data. All fastq files need to be homogeneous on their extension name
OUTPUT	output *path* directory

Warning: For FASTQ, naming convention accepted by CulebrONT is *NAME.fastq.gz* or *NAME.fq.gz* or *NAME.fastq* or *NAME.fq*

All fastq files have to be homogeneous on their extension and can be compressed or not.

Reference fasta file need a fasta or fa extension uncompressed.

2. Chose assemblers, polisher and correctors

Activate/deactivate assemblers, polishers and correctors as you wish. Feel free to activate only assembly, assembly+polishing or assembly+polishing+correction.

Note: If you are working on prokaryote, is recommendated to activate CIRCULAR steps

Example:

```
ASSEMBLY :
    CANU : False
    FLYE : True
    MINIASM : True
    RAVEN : False
    SMARTDENOVO :  False
    SHASTA : False

POLISHING :
    RACON : True

CIRCULAR : False
```

(continues on next page)

(continued from previous page)

```
CORRECTION :
    NANOPOLISH : True
    MEDAKA : False

FIXSTART : False
```

3. Chose quality tools

With CulebrONT you can use several quality tools to check assemblies.

- If BUSCO or QUAST are used, every fasta generated on the pipeline will be used with them.
- If BLOBTOOLS, ASSEMBLYTICS, WEESAM and KAT are activated only the last draft generated on the pipeline will be used.
- KAT quality tool can be activate but Illumina reads are mandatory in this case. These reads can be compressed or not.

```
# BUSCO and QUAST will be launched on all activated steps (ASSEMBLY,␣
↪POLISHING, CORRECTION)
QUALITY:
    BUSCO: True
    QUAST: True
#### Others quality tools are lauched only in last assemblies
    WEESAM: False
    BLOBTOOLS: False
    ASSEMBLYTICS: False
#### Others quality softs but illumina reads are required
    KAT: True
```

Alignment of various assemblies **for small genomes (<10-20Mbp)** is also possible by using Mauve.

- If you want to improve alignment with MAUVE on circular molecules is recommended to activate *Fixstart* step.
- Only activate MAUVE if you have more than one sample and more than one quality step.

```
#### Alignment of the various assemblies derived from a fastq file for␣
↪small genomes (<10-20Mbp);
MSA:
    MAUVE: False
```

4. Parameters for some specific tools

You can manage tools parameters on the params section on `config.yaml` file.

Specifically to `Racon`:

- Racon can be launch recursively from 1 to 9 rounds. 2 or 3 are recommended.

Specifically to `Medaka`:

- If 'MEDAKA_TRAIN_WITH_REF' is activated, Medaka launchs training using the reference found in 'DATA/REF' param. Medaka does not take into account other medaka model parameters.
- If 'MEDAKA_TRAIN_WITH_REF' is deactivated, Medaka does not launch training but uses instead the model provided in 'MEDAKA_MODEL_PATH'. Give to CulebrONT path of medaka model in order to correct assemblies. This parameter could not be empty.

Important: Medaka models can be downloaded from the medaka repository. You need to install `git lfs` (see documentation here https://git-lfs.github.com/) to download largest files before `git clone https://github.com/nanoporetech/medaka.git\`.

Here you find standard parameters used on CulebrONT. Feel free to adapt it to your requires.

```
params:
    #### ASSEMBLY
    MINIMAP2:
        PRESET_OPTION: 'map-ont' # -x minimap2 preset option is map-pb by␣
↪default (map-pb, map-ont etc)
    FLYE:
        OPTIONS: ''
    CANU:
        MAX_MEMORY: '50G'
        OPTIONS: '-fast'
    SMARTDENOVO:
        KMER_SIZE: 16
        OPTIONS: '-J 5000'
    SHASTA:
        MEM_MODE: 'filesystem'
        MEM_BACKING: 'disk'
        OPTIONS: '--Reads.minReadLength 0'


    #### CIRCULAR
    CIRCLATOR:
        OPTIONS: ''


    #### POLISHING
    RACON:
        RACON_ROUNDS: 2                 #1 to 9


    #### CORRECTION
    CORRECTION_MAKERANGE:
        SEGMENT_LEN: '50000'            # segment length to split assembly␣
↪and correct it  default=50000
        OVERLAP_LEN: '200'              # overlap length between segments ␣
↪default=200

    NANOPOLISH:
        OPTIONS: ''
```

(continues on next page)

(continued from previous page)

```
    MEDAKA:
        MEDAKA_TRAIN_WITH_REF: False    # if 'MEDAKA_TRAIN_WITH_REF' is␣
↪True, training uses reference found in DATA REF param.

        # Medaka does not take in count other parameters below if MEDAKA_
↪TRAIN_WITH_REF is TRUE.
        MEDAKA_MODEL_PATH: './Data-Xoo-sub/medaka-models/r941_min_high_
↪g303_model.hdf5' # if empty this param is forgotten.
        MEDAKA_FEATURES_OPTIONS: '--batch_size 10 --chunk_len 100 --chunk_
↪ovlp 10'
        MEDAKA_TRAIN_OPTIONS: '--batch_size 10 --epochs 500 '
        MEDAKA_CONSENSUS_OPTIONS: '--batch 200 '


    #### QUALITY
    BUSCO:
        DATABASE : './Data-Xoo-sub/bacteria_odb10'
        MODEL : 'genome'
        SP : ''                         #--augustus-specie parametter on␣
↪busco

    QUAST:
        GFF: ''
        OPTIONS : '--large'

    DIAMOND:
        DATABASE: './Data-Xoo-sub/testBacteria.dmnd'

    MUMMER:
        MINMATCH : 100                  # is -l option with default 20 on␣
↪MUMMER
        MINCLUSTER: 500                 # is -c option with default 65 on␣
↪MUMMER

    ASSEMBLYTICS:
        UNIQUE_ANCHOR_LEN: 10000
        MIN_VARIANT_SIZE: 50
        MAX_VARIANT_SIZE: 10000
```

Warning: Please check documentation of each tool and make sure that the settings are correct!

How to run the workflow ?

Command line

Before lauch culebrONT please be sure you have already modified the `config.yaml` file as was explained on *1. Providing data*

This is the recommended Snakemake command line to run CulebrONT:

```
snakemake --nolock --use-conda --use-singularity --singularity-args '--
↪bind $HOME' --cores -p -s Snakefile --latency-wait 6000000 --keep-going -
↪-restart-times 0 --rerun-incomplete --configfile config.yaml --conda-
↪prefix $PWD/build_conda_envs
```

`config.yaml` file is give to Snakemake by the argument `--configfile`

To launch CulebrONT, you should use the parameters `--use-singularity` and `--use-conda`. Please don't forget to export conda on your $PATH.

Snakemake compiles in each output directory conda environment. To avoid this, please use `--conda-prefix /path/to/build_conda_env` on snakemake command line.

Bind mount disks to singularity environment by using `--singularity-args '--bind $YOURMOUNTDISK'`. It allows to detect others disk inside of the singularity container. $YOURMOUNTDISK corresponds to mount disk, it could be $HOME or another disk path.

Note: For others snakemake arguments, please check documentation https://snakemake.readthedocs.io/en/v5.11.0/executing/cli.html#all-options

---

Cluster execution

This is a typical launcher for using CulebrONT on a SLURM cluster. You have to adapt it for the configuration of your favorite one. You can use wrappers or profiles.

wrappers

A `slurm_wrapper.py` script is available on CulebrONT projet to manage resources from your cluster configuration (from cluster_config.yaml file). This is the easier way to know what is running on cluster and to adapt resources for every job. Take care, this cluster_config.yaml file is becoming obsolete on latest Snakemake versions.

```
#!/bin/bash
#SBATCH --job-name culebrONT
#SBATCH --output slurm-%x_%j.log
#SBATCH --error slurm-%x_%j.log

module load system/singularity/3.3.0
module load system/python/3.7.2

snakemake --unlock

# SLURM JOBS WITH USING WRAPPER
snakemake --nolock --use-conda --use-singularity --cores -p -s Snakefile --
↪latency-wait 60000000 --keep-going --restart-times 0 --rerun-incomplete -
↪-configfile config.yaml --cluster "python3 slurm_wrapper.py config.yaml␣
↪cluster_config.yaml" --cluster-config cluster_config.yaml --cluster-
↪status "python3 slurm_status.py"
```

profiles

Optionally is possible to use Profiles in order to run CulebrONT on HPC cluster. Please follow the recommendations found on the SnakeMake profile github.

```
#!/bin/bash
#SBATCH --job-name culebrONT
#SBATCH --output slurm-%x_%j.log
#SBATCH --error slurm-%x_%j.log

module load system/singularity/3.3.0
module load system/python/3.7.2

snakemake --unlock

# USING PROFILES
snakemake --nolock --use-singularity --use-conda --cores -p -s Snakefile --
↪configfile config.yaml --latency-wait 60000000 --keep-going --restart-
↪times 0 --rerun-incomplete --cluster-config cluster_config.yaml --
↪profile slurm-culebrONT
```

Note: For others snakemake cluster arguments, please check documentation https://snakemake.readthedocs.io/en/stable/executing/cluster.html

In any case, this launcher can be submitted to the SLURM queue typing:

```
sbatch submit_culebront.sh
```

Output on CulebrONT

The architecture of CulebrONT output is designed as follows:

```
OUTPUT_CULEBRONT_CIRCULAR/
├── SAMPLE-1
│   ├── AGGREGATED_QC
│   │   ├── DATA
│   │   ├── MAUVE_ALIGN
│   │   └── QUAST_RESULTS
│   ├── ASSEMBLERS
│   │   ├── CANU
│   │   │   ├── ASSEMBLER
│   │   │   ├── CORRECTION
│   │   │   ├── FIXSTART
│   │   │   ├── POLISHING
│   │   │   └── QUALITY
│   │   ├── FLYE
│   │   │   ├── ...
│   │   ├── MINIASM
│   │   │   ├── ...
```

(continues on next page)

(continued from previous page)

```
│   │   ├── RAVEN
│   │   │   ├── ...
│   │   ├── SHASTA
│   │   │   ├── ...
│   │   └── SMARTDENOVO
│   │   │   ├── ...
│   ├── DIVERS
│   │   └── FASTQ2FASTA
│   ├── LOGS
│   └── REPORT
└── FINAL_REPORT
├── SAMPLE-2 ...
```

Report

CulebrONT generates a useful report containing, foreach fastq, a summary of interesting statistics. Please discover an ... and enjoy !!

Important: To visualise the report created by CulebrONT, transfer the folder `FINAL_RESULTS` on your local computer and open it on a navigator.

Citation

@Authors:

Julie Orjuela (IRD), Aurore Comte(IRD), Sébastien Ravel(CIRAD), Florian Charriat(INRAE), Tram Vi(IRD, AGI), Francois Sabot(IRD) and Sébastien Cunnac(IRD).

Useful notes

Before launching CulebrONT, you could base-calling of arbitrarily multiplexed libraries across several Minion runs with sequencing quality control and gather the output files by genome for subsequent steps. For that use https://github.com/vibaotram/baseDmux.

Thanks

Thanks to Ndomassi Tando (i-Trop IRD) by administration support.

The authors acknowledge the IRD i-Trop HPC (South Green Platform) at IRD Montpellier for providing HPC resources that have contributed to this work. https://bioinfo.ird.fr/ - http://www.southgreen.fr

Thanks to Yann Delorme for this beautiful logo https://nimarell.github.io/resume

License

Licencied under CeCill-C (http://www.cecill.info/licences/Licence_CeCILL-C_V1-en.html) and GPLv3 Intellectual property belongs to IRD and authors.

Frequently Asked Questions

CulebrONT report does not show Quast results

- To observe results on CulebrONT final report you have to transfer the whole of repertory. `FINAL_RESULTS` contains QUAST directory and also the `snakemake-report.htlm` who is generated by snakemake argument –report.

Error validating model from –model argument

- This error is obtained if you are not bind your singularity. It's necessary to launch snakemake command line by using `--singularity-args '--bind $HOME'` arguments.

How much space does it take to install the CulebrONT dependencies?

- build_conda_envs takes 2.1G and build singularity about 4.9G (medaka-gpu-1.2.simg takes 2.1G)

Recommended Snakemake command line to launch CulebrONT

```
snakemake --nolock --use-conda --use-singularity --singularity-args '--
↪bind $HOME' --cores -p -s Snakefile --latency-wait 6000000 --keep-going -
↪-restart-times 0 --rerun-incomplete --configfile config.yaml --conda-
↪prefix $PWD/build_conda_envs
```

- Snakemake compiles in each output directory conda environments. To avoid this, please use `--conda-prefix /path/to/build_conda_env` on snakemake command line. This could be practical on HPC cluster install.
- Bind mount disks to singularity environment by using `--singularity-args '--bind $HOME'`. This allows to detect others disks on the singularity container.

The conda command is not available in the shell /bin/bash that will be used by Snakemake

Please check conda installation. On i-Trop cluster you can add it on sbatch script. you have to adapt it to your favorite cluster.

```
module load system/Miniconda3/1.0

env=/home/$(whoami)/.conda/envs/snakemake
[ ! -d $env ] && echo -e "## [$(date) - culebrONT]\t Creating conda␣
↪environment for snakemake" && conda env create -f envs/environment.yaml -
↪n snakemake

source activate snakemake
```

** QUAST takes to much time to run**

Add `--large` to quast options

Problem building conda environments

CreateCondaEnvironmentException: Could not create conda environment from /shared/ifbstor1/home/jorjuela/softs/CulebrONT_pipeline/envs/nanopolish_minimap2_samtools_seqtk.yaml: Collecting package metadata (repodata.json): ...working... done Solving environment: ...working... Building graph of deps: 0%| | 0/8 [00:00<?, ?it/s] Examining minimap2=2.17: 0%| | 0/8 [00:00<?, ?it/s] Examining seqtk=1.3: 12%| | 1/8 [00:00<00:00, 138.72it/s] Examining @/linux-64::__archspec==1=x86_64: 25%| | 2/8 [00:00<00:00, 149.47it/s] Examining samtools=1.10: 38%| | 3/8 [00:00<00:00, 220.81it/s] Examining python=3.7: 50%| | 4/8 [00:00<00:00, 169.27it/s] Examining nanopolish=0.13.2: 62%| | 5/8 [00:01<00:00, 3.16it/s] Examining nanopolish=0.13.2: 75%| | 6/8 [00:01<00:00, 3.79it/s] Examining @/linux-64::__unix==0=0: 75%| | 6/8 [00:01<00:00, 3.79it/s] Examining @/linux-64::__unix==0=0: 88%| | 7/8 [00:01<00:00, 3.56it/s] Examining @/linux-64::__glibc==2.17=0: 88%| | 7/8 [00:01<00:00, 3.56it/s]

test use mamba instead of conda `--conda-frontend mamba`