

# Supplementary information to CulebrONT pipeline

Orjuela J., Comte A., Ravel S., Charriat F., Vi T., Sabot F., Cunnac S.

July 23, 2022

## 1 CulebrONT's architecture

CulebrONT is a snakemake pipeline originally created to assemble ONT reads, but also working on PB, that can also polish primary assemblies and provide quality metrics, in order to facilitate comparison of assembly methods. Using CulebrONT, workflows can be easily generated to assemble genome from various organisms, from bacteria to more complex eukaryots. Dedicated and optional steps are included to take into account the specific features of procaryotic genomes assembly, *i.e.* circularization and definition of the origin of replication.

In order to build a pipeline, the user customizes mainly a file in YAML (Yet Another Markup Language) format: *config.yaml*. Two other files may be necessary to install dependencies *tools\_path.yaml* and manage resources on a HPC with a job scheduler environment *cluster\_config.yaml*. CulebrONT uses the snakemake architecture in order to run a relevant set of rules, each one being executed into a dedicated virtual environment.

### 1.1 *config.yaml*

The main workflow parameters determining the assembly strategy are stored and read by CulebrONT from *config.yaml* (see figure 1.A). This file provides the path to the fastq (required, plain or gzipped), fast5 and reference genome (if any) files. For each step (assembly, polishing or correction), the user can choose which tools, including quality evaluation tools, must be launched and can define custom options (or default) for these programs. This allows users to run different workflows on exactly the same dataset, while recording tools versions and parameters.

### 1.2 *tools\_path.yaml*

CulebrONT uses singularity to setup dependencies of each workflow (see figure 1.B). Pre built images are available on the i-Trop IRD platform server [https://itrop.ird.fr/culebront\\_utilities/singularity\\_build/](https://itrop.ird.fr/culebront_utilities/singularity_build/). It is also possible to build singularity images on your local or HPC machines, but root access is necessary. Dedicated singularity recipes can be found on our github repository ([https://github.com/SouthGreenPlatform/CulebrONT\\_pipeline](https://github.com/SouthGreenPlatform/CulebrONT_pipeline)). These *Singularity recipes* containers allows an easy installation of CulebrONT dependencies tools. Alternatively, CulebrONT also allows the use of Environment Modules usually found on clusters.

The *tools\_path.yaml* file specifies the paths of the singularity containers or of the available module environments.

### 1.3 *cluster\_config.yaml*

Within CulebrONT, users can take full advantage of Snakemake for workflow execution on a cluster or on the cloud.

The *cluster\_config.yaml* file defines the requested HPC resources (memory, CPU number, partitions, CPU/GPU access) for executing rules on a cluster with a job scheduler, and can be adapted by the user or system administrators (see figure 1.C). If needed, execution through command-line cluster wrappers provide additional controls and customization possibilities to the user. An easy-to-use Snakemake profile method is described on our documentation page. In the absence of a cluster setup, the workflow will be executed locally.

A. <i>config.yaml</i>		B. <i>tools_path.yaml</i>	
path to data	<pre>DATA: FASTQ: 'path/to/fastq' FAST5: 'path/to/fast5/' ILLUMINA: 'path/to/illumina/' GENOME_SIZE: '4.8m' REF: 'path/to/ref/ref.fasta' OUTPUT: 'path/to/output/'</pre>	Available singularity or module environment	<pre>SINGULARITY: REPORT: '.Containers/Singularity.report.sif' TOOLS: '.Containers/Singularity.culebront_tools.sif'  ENVMODULE: R: "R/4.0.2" QUAST: "quast/5.0.2" MAUVE: "mauve/2.4.0" SHASTA: "shasta/0.1.0" ASSEMBLYTICS: "assemblytics/1.0" MEDAKA: "medaka-gpu/0.10.0"</pre>
activate pipeline tools	<pre>ASSEMBLY: CANU: True FLYE: True MINIASM: True RAVEN: True SMARTDENOVO: True SHASTA: True  POLISHING: RACON: True CIRCULAR: True CORRECTION: NANOPOLISH: True MEDAKA: True FIXSTART: True</pre>		
activate quality tools	<pre>QUALITY: BUSCO: True QUAST: True FLAGSTATS: True BLOOTOOLS: True ASSEMBLYTICS: True KAT: True  MSA: MAUVE: True</pre>		
adapt params	<pre>params: MINIMAP2: PRESET_OPTION: 'map-ont' FLYE: OPTIONS: '' CANU: MAX_MEMORY: '15G' OPTIONS: '-fast'</pre>	Adapt threads number, GPU or CPU resources and partitions on cluster	<pre>__default__: cpus-per-task: 4 ntasks: 1 mem-per-cpu: '2' partition: "normal" output: 'logs/stdout/{rule}/{wildcards}' error: 'logs/error/{rule}/{wildcards}'  run_canu: cpus-per-task: 8 mem-per-cpu: '4' partition: "long"  run_medaka_train: cpus-per-task: 8 mem-per-cpu: '4' partition: "gpu"</pre>
C. <i>cluster_config.yaml</i>			

Figure 1: Configuration files setup defining an instance of the workflow.

## 2 How to run CulebrONT?

A comprehensive documentation can be found online <https://culebront-pipeline.readthedocs.io/en/latest/> and a test dataset is available on the i-Trop server [https://itrop.ird.fr/culebront\\_utilities/](https://itrop.ird.fr/culebront_utilities/)

### 2.1 Building pipelines from *config.yaml*

We present here some examples of modular pipelines build using the *config.yaml* file (figure 2). Users can be interested to test the best assembly pipeline on a bacterial reference strain before launching it on multiple related strains, for instance. If an eukaryotic organism with a large genome size ( $\geq 100\text{Mb}$  or more) is targeted, starting with testing all assemblers and checking quality of raw assemblies before relaunching CulebrONT with polishing and correction with only the best assembler(s) is more relevant. The user can thus choose optional tools for each integrated step.

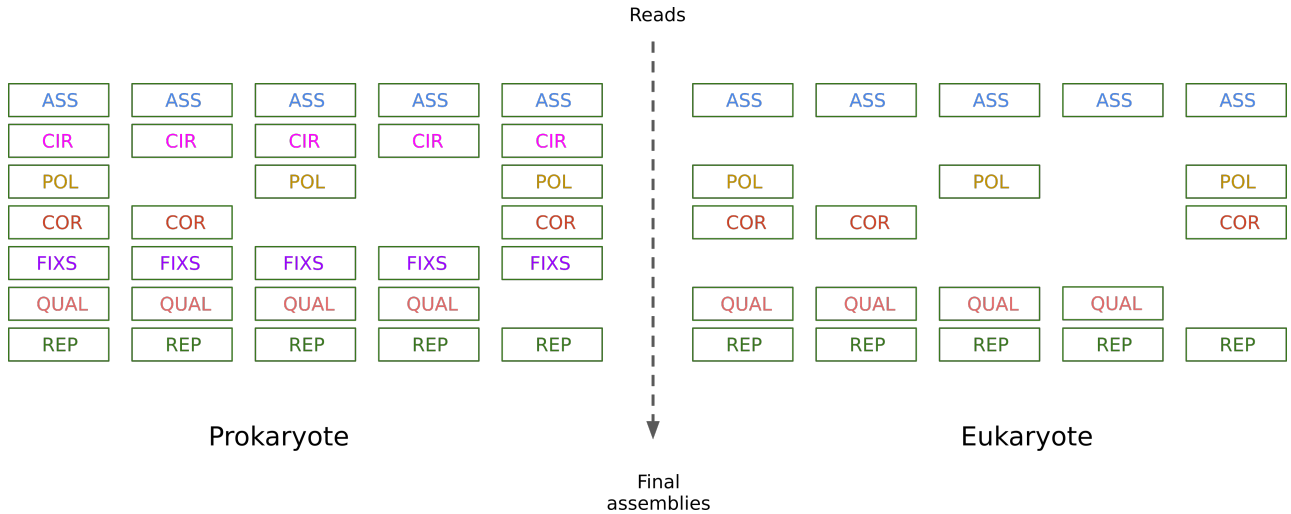


Figure 2: Several possibilities to build a workflow. ASS: assembly, CIR: circularisation, POL: polishing, COR: correction, FIXS: fixstart, QUAL: quality control, REP: final report. For each step, multiple tools are available.

## 2.2 Building a pipeline for prokaryotes or eukaryotes

**Assembly:** CulebrONT currently includes six recent and community-validated assemblers (Figures 4 and 3, blue squares): Flye[1], Canu[2], Miniasm[3] + Minipolish[4], Shasta [5], Smartdenovo[6] and Raven[7]. Tools versions for each release can be found in the CulebrONT documentation.

- Canu[2] performs assembly by using corrected and trimmed reads. Its assembly strategy uses an overlap-layout-consensus (OLC) approach.
- Flye[1] generates the concatenation of multiple disjoint genomic segments called disjointigs to build a repeat graph. Reads are mapped to this repeat graph to resolve conflicts (unbridged repeats) and output contigs.
- Miniasm[3] performs the layout step of OLC. Read overlapping is performed separately with Minimap2[3].
- Raven[7] also uses an OLC approach. The overlapping step is similar to Miniasm and performed by Minimap2. The initial consensus step uses Racon[8]. Layout step removes spurious overlaps from the graph, improving contiguity.
- Smartdenovo[6] performs read overlapping, rescues missing overlaps, identifies low-quality regions and produces a consensus.
- Shasta[5] is a computationally efficient assembler, relying on a reduced representation of marker k-mers used to find overlaps and to build an assembly graph.

**Polishing:** This step is ensured by Racon[8] (Figures 4 and 3, yellow squares). Racon corrects raw contigs generated by raw assembly methods using the original ONT or PB reads. The user can specify a number of Racon rounds (constrains from 1 to 9 rounds) and CulebrONT will recursively execute them (generally between 2 to 4 iterations are commonly done). Polishing from assembled pseudomolecules obtained using Miniasm is automatically performed by Racon, also included in minipolish.

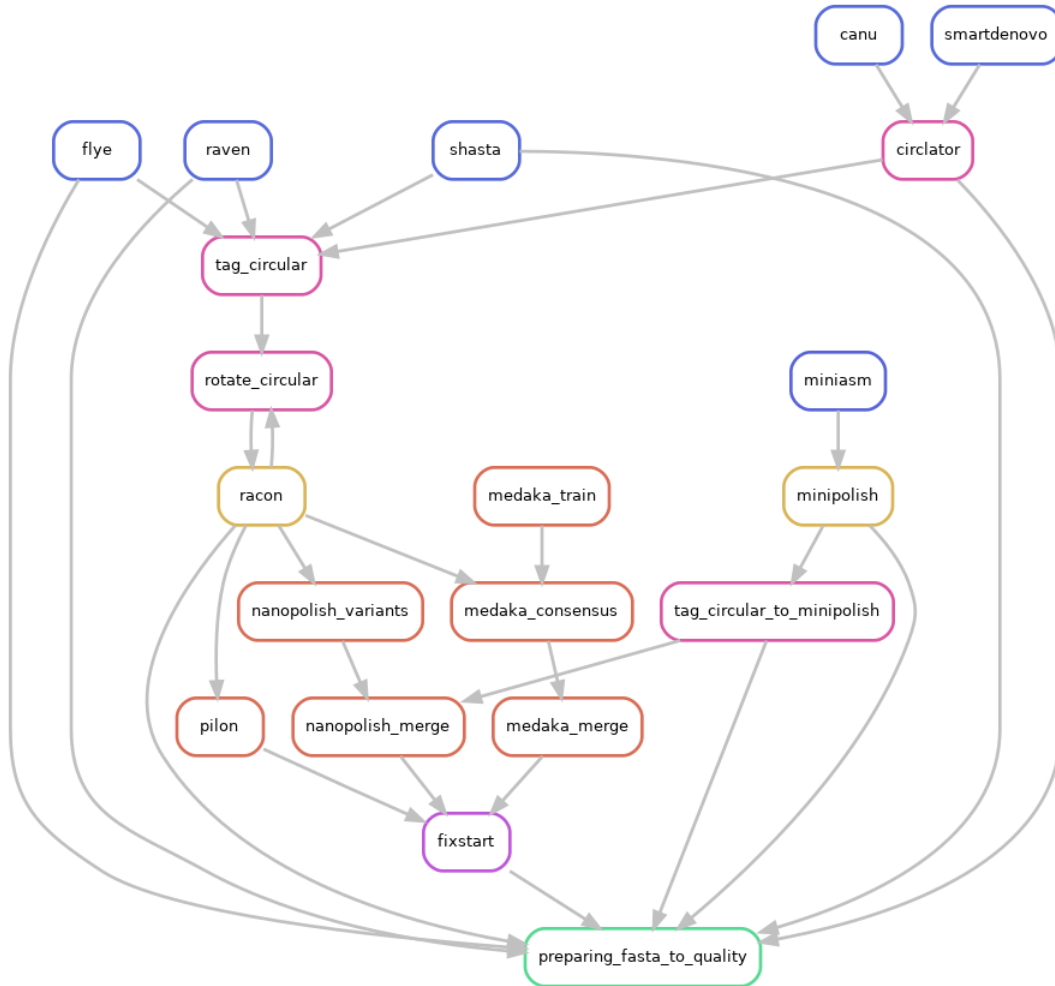


Figure 3: Circular option on *config.yaml* file is activated

**Correction:** Correction can improve the consensus sequence for a draft or polished ONT-based genome assembly. Community validated tools Nanopolish[9] and Medaka<sup>1</sup> are included (Figures 3 and 4, orange squares). Medaka uses neural networks applied to a pileup of individual sequencing reads in fastq against a draft assembly. The model training can be performed using CulebrONT or provided externally. Correction by segments of the assembled molecules is managed by CulebrONT, for both Nanopolish and Medaka, in a parallel way, improving speed.

Pilon[10] has been recently included and can be activated in order to correct assemblies using Illumina short reads.

Even if prokaryotic genomes are smaller and less repetitive than eukaryotic ones, **Circularisation** represents a specific dimension of the assembly task. If an assembled molecule, for example a bacterial chromosome or a plasmid, is identified as circular, this molecule is tagged and will be specially treated in the pipeline (see figure 3, pink squares). If the circular step is activated, the `--plasmids` option on Flye ( $\leq v2.9$ ) will be automatically used. For Canu and Smartdenovo, Circlator[11] is used to circularise assemblies. Circularisation for Miniasm is performed by Minipolish[4], which first polishes Miniasm graph assembly using Racon and then handles circularisation. In CulebrONT, for others assemblers (not miniasm), tagging and rotation of circular molecule before each polishing step is implemented. The starting position of circular contigs is arbitrarily shifted between polish-

<sup>1</sup><https://github.com/nanoporetech/medaka>

ing rounds. The circular status of Miniasm, Raven and Shasta contigs is determined by inspecting assemblers GFA files, or a log file for Flye and Circlator, in a dedicated *tag\_circular* step, where only circular fasta sequences are tagged.

The **Fixstart** rule uses the fasta file generated after the most downstream step of the pipeline. It shifts the start position of circular sequences so that the origin of replication of the contig coincides with the start of the ortholog of the *dnaA* gene (if found). This is useful to harmonize the origin of bacterial genomes in a multiple alignment.

The circular option can be *True* or *False* in the *config.yaml* file. The Figure 4 presents a possible pipeline without the circular step, for eukaryotes for instance.

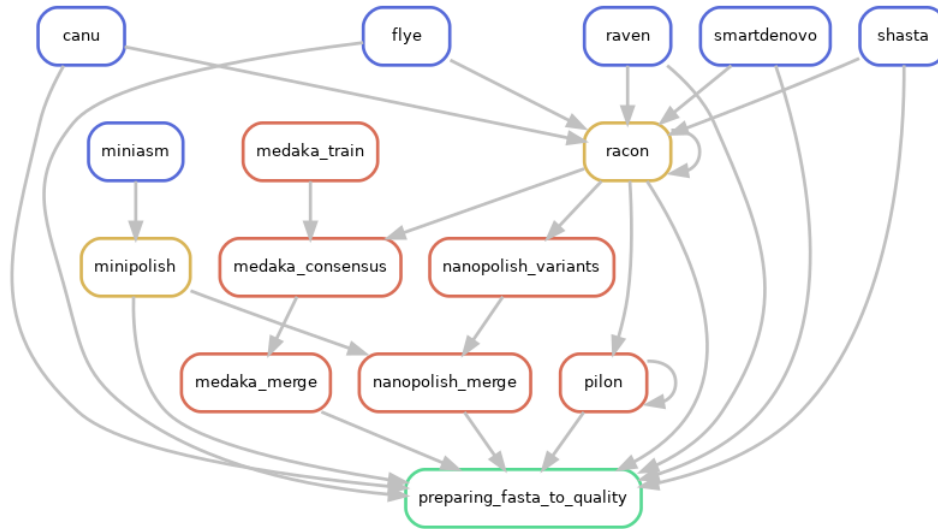


Figure 4: Circular option on *config.yaml* file is deactivated

### 2.3 Quality control

A wide variety of quality control tools is implemented in CulebrONT to check accuracy of a given assembly (all optional). *Busco*[12] evaluates genome completeness and accuracy by comparison of predicted genes with a specified database of conserved genes. Various metrics, such as the number of contigs, length of the longest contig, N50, number of predicted genes, numbers of potential misassemblies are computed using *Quast*[13]. If *Busco* or *Quast* are selected, their statistics will be computed for all assemblies generated by the pipeline, regardless of their stage in the workflow. Their statistics are summarized in the CulebrONT final HTML report. The complete output of these tools can also be explored in the output directory.

Supplemental tools are also included: *blobtools*[14] to detect contamination, *assemblytics*[15] to compare contiguity of the assemblies against a reference genome, *KAT*[16] and *Merqury*[17] to explore k-mers frequencies and check possible contamination, samtools FLAGSTATS to compute remapping percentage over assemblies, and *Mauve*[18] to generate a multiple-alignment of the alternative assemblies (for small bacterial genomes). Only the fasta file generated after the most downstream step of the pipeline is analyzed by these optional tools. If selected, their statistics are also summarized in the CulebrONT final HTML report.

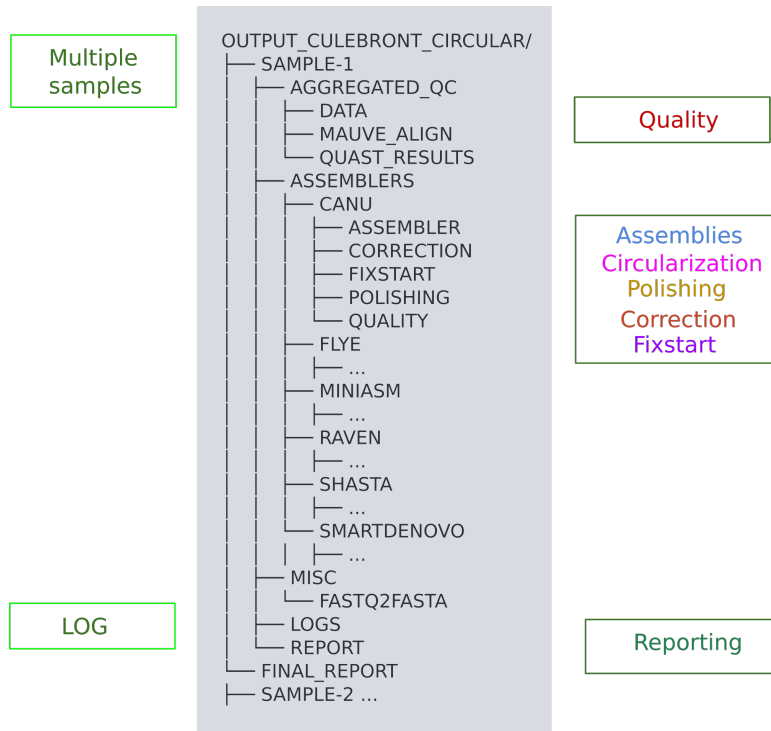


Figure 5: Output folder architecture

## 2.4 Output and reporting

CulebrONT generates an output directory with a specific architecture (Figure 5). For each analysed sample, the output directory contains subdirectories named after selected steps in the *config.yaml* file (assembly, circularization, polishing, correction, fixstart, and quality control). In addition, a **LOG** directory contains all logs created. In the **FINAL\_REPORT** folder, CulebrONT generates an HTML report, summarizing statistics from the various steps of the pipeline and including a record of the configuration parameters used in the pipeline, the version of the software tools and the statistics about computing time and resources. An example of the report generated by the test dataset can be found here: [https://itrop.ird.fr/culebront\\_utilities/FINAL\\_REPORT/CulebrONT\\_report.html](https://itrop.ird.fr/culebront_utilities/FINAL_REPORT/CulebrONT_report.html).

## 2.5 Installation using a PyPi package

CulebrONT is available as a Pypi package <https://pypi.org/project/culebrONT/>. This Python API allows users easy install CulebrONT in local or HPC machines <https://culebront-pipeline.readthedocs.io/en/latest/>.

## 2.6 Conclusion

CulebrONT can be used in many experiments and on many samples, and in addition is a FAIR product :

- Open source: You can find it on github with a licence under CeCill-C ([http://www.cecill.info/licences/Licence\\_CeCILL-C\\_V1-en.html](http://www.cecill.info/licences/Licence_CeCILL-C_V1-en.html)) and GPLv3. You can modify and redistribute it.
- Modulable: All steps to obtain the best quality assembly are included, and can be activated (or not) according to your needs. All the most commonly used tools in the community are

integrated, as well as various state-of-the-art quality control tools. The possibility to use containers makes it easy to change and control tools version.

- Scalable: most eukaryote or prokaryote genome can be assembled and circular molecules are handled accordingly. You just need to fill in a simple text-based configuration file to build a personalised pipeline. The limits are imposed by the hardware, not the code.
- Intuitive: in order to quickly give the user an idea of the best final assembly, a well structured output directory and log files are provided as well as a report with statistics and images of the most important steps.
- Traceable: users can reproduce the analysis easily using the report and config.yaml, and the HTML report will provide all information about softwares version, original data, QC, and options used.

## 2.7 Related pipeline of interest

BaseDMux<sup>2</sup> is another snakemake pipeline developed in collaboration with the CulebrONT projet. It allows basecalling, demultiplexing and filtering ONT data in a flexible way before assemblies. Its output can be used directly as input for CulebrONT.

## 2.8 Competing interests

The authors declare no competing interests.

## References

- [1] Kolmogorov, M., Yuan, J., Lin, Y., Pevzner, P.A.: Assembly of long, error-prone reads using repeat graphs. *Nature Biotechnology* **37**(5), 540–546 (2019). doi:10.1038/s41587-019-0072-8
- [2] Koren, S., Walenz, B.P., Berlin, K., Miller, J.R., Bergman, N.H., Phillippy, A.M.: Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome Research* **27**(5), 722–736 (2017). doi:10.1101/gr.215087.116. <http://genome.cshlp.org/content/27/5/722.full.pdf+html>
- [3] Li, H.: Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics* **32**(14), 2103–2110 (2016). doi:10.1093/bioinformatics/btw152. <https://academic.oup.com/bioinformatics/article-pdf/32/14/2103/19567911/btw152.pdf>
- [4] Wick, R.R., Holt, K.E.: Benchmarking of long-read assemblers for prokaryote whole genome sequencing [version 4; peer review: 4 approved]. *F1000Research* **8**(2138) (2021). doi:10.12688/f1000research.21782.4
- [5] Shafin, K., Pesout, T., Lorig-Roach, R., Haukness, M., Olsen, H.E., Bosworth, C., Armstrong, J., Tigyi, K., Maurer, N., Koren, S., Sedlazeck, F.J., Marschall, T., Mayes, S., Costa, V., Zook, J.M., Liu, K.J., Kilburn, D., Sorensen, M., Munson, K.M., Vollger, M.R., Monlong, J., Garrison, E., Eichler, E.E., Salama, S., Haussler, D., Green, R.E., Akeson, M., Phillippy, A., Miga, K.H., Carnevali, P., Jain, M., Paten, B.: Nanopore sequencing and the shasta toolkit enable efficient de novo assembly of eleven human genomes. *Nature Biotechnology* **38**(9), 1044–1053 (2020). doi:10.1038/s41587-020-0503-6
- [6] Liu, H., Wu, S., Li, A., Ruan, J.: Smartdenovo: A de novo assembler using long noisy reads. <https://www.preprints.org/> (2020). doi:10.20944/preprints202009.0207.v1. <https://www.preprints.org/manuscript/202009.0207/v1>

---

<sup>2</sup><https://github.com/vibaotram/baseDmux>

- [7] Vaser, R., Šikić, M.: Raven: a de novo genome assembler for long reads. *bioRxiv* (2020). doi:10.1101/2020.08.07.242461. <https://www.biorxiv.org/content/early/2020/08/10/2020.08.07.242461.full.pdf>
- [8] Vaser, R., Sović, I., Nagarajan, N., Šikić, M.: Fast and accurate de novo genome assembly from long uncorrected reads. *Genome research* **27**(5), 737–746 (2017). doi:10.1101/gr.214270.116.28100585[pmid]
- [9] Loman, N.J., Quick, J., Simpson, J.T.: A complete bacterial genome assembled de novo using only nanopore sequencing data. *Nature Methods* **12**(8), 733–735 (2015). doi:10.1038/nmeth.3444
- [10] Walker, B.J., Abeel, T., Shea, T., Priest, M., Abouelliel, A., Sakthikumar, S., Cuomo, C.A., Zeng, Q., Wortman, J., Young, S.K., Earl, A.M.: Pilon: An integrated tool for comprehensive microbial variant detection and genome assembly improvement. *PLoS ONE* **9**(11), 112963 (2014). doi:10.1371/journal.pone.0112963
- [11] Hunt, M., Silva, N.D., Otto, T.D., Parkhill, J., Keane, J.A., Harris, S.R.: Circlator: automated circularization of genome assemblies using long sequencing reads. *Genome Biology* **16**(1), 294 (2015). doi:10.1186/s13059-015-0849-0
- [12] Simão, F.A., Waterhouse, R.M., Ioannidis, P., Kriventseva, E.V., Zdobnov, E.M.: BUSCO: assessing genome assembly and annotation completeness with single-copy orthologs. *Bioinformatics* **31**(19), 3210–3212 (2015). doi:10.1093/bioinformatics/btv351. <https://academic.oup.com/bioinformatics/article-pdf/31/19/3210/17086320/btv351.pdf>
- [13] Gurevich, A., Saveliev, V., Vyahhi, N., Tesler, G.: QUAST: quality assessment tool for genome assemblies. *Bioinformatics* **29**(8), 1072–1075 (2013). doi:10.1093/bioinformatics/btt086
- [14] Laetsch, D., Blaxter, M.: Blobtools: Interrogation of genome assemblies [version 1; peer review: 2 approved with reservations]. *F1000Research* **6**(1287) (2017). doi:10.12688/f1000research.12232.1
- [15] Nattestad, M., Schatz, M.C.: Assemblytics: a web analytics tool for the detection of variants from an assembly. *Bioinformatics* **32**(19), 3021–3023 (2016). doi:10.1093/bioinformatics/btw369
- [16] Mapleson, D., Garcia Accinelli, G., Kettleborough, G., Wright, J., Clavijo, B.J.: KAT: a k-mer analysis toolkit to quality control NGS datasets and genome assemblies. *Bioinformatics* **33**(4), 574–576 (2017)
- [17] Rhie, A., Walenz, B.P., Koren, S., Phillippy, A.M.: Merqury: reference-free quality, completeness, and phasing assessment for genome assemblies. *Genome Biology* **21**(1), 245 (2020). doi:10.1186/s13059-020-02134-9
- [18] Darling, A.C.E., Mau, B., Blattner, F.R., Perna, N.T.: Mauve: multiple alignment of conserved genomic sequence with rearrangements. *Genome research* **14**(7), 1394–1403 (2004). doi:10.1101/gr.2289704.15231754[pmid]