

User Manual

LRSDAY: Long-read Sequencing Data Analysis for Yeasts

Release v1.7.1

(2023-01-28)

Author Contact:

Jia-Xing Yue (岳家兴)

Email: [yuejiaxing@gmail.com](mailto:yuejiaxing@gmail.com)

GitHub: [yjx1217](https://github.com/yjx1217)

Twitter: [iAmphioxus](https://twitter.com/iAmphioxus) [evomicslab](https://twitter.com/evomicslab)

Website: [http://www.evomicslab.org](http://www.evomicslab.org/)

Table of Contents

INTRODUCTION 1

Background 1

Overview of the LRSDAY workflow 2

Comparison with other methods 3

Experimental Design 4

ANTICIPATED RESULTS 9

Limitations and potential adaptation 9

Expected improvements 9

CITATIONS 10

MATERIALS 10

Hardware, operating system and network 10

Software or library requirements 11

Input data 11

Example data 12

PROCEDURE 13

Download, install and configure LRSDAY 13

Run LRSDAY with the testing example 16

REFERENCES 37

APPENDIX 39

Appendix 1: Pre-shipped supporting data for LRSDAY 39

Appendix 2: Tips for adapting LRSDAY to other eukaryotic organisms 40

# INTRODUCTION

### Background

Twenty years ago, the genome sequence of the budding yeast *Saccharomyces cerevisiae* was published1. As the first complete eukaryotic genome ever sequenced, this marked a major scientific milestone in biology. Since then, the genomes of many model and non-model organisms have been sequenced, with this process accelerating after the emergence of next-generation sequencing (NGS) technologies. Despite the notably improved throughputs, NGS technologies suffer from the limitation of short reads and usually result in highly fragmented genome assemblies containing numerous gaps and local mis-assemblies. The recently developed long-read sequencing technologies represented by Pacific Biosciences (PacBio) and Oxford Nanopore offer compelling alternatives to overcome such hurdles, producing high-quality genome assemblies with substantially improved continuity and accuracy. Although initially tested in microbial genome sequencing, their recent applications in complex mammalian and plant genomes also achieved high-quality results2–6. With such new sequencing technologies, challenging genomic regions with enriched repetitive elements, strongly biased GC-content, or complex structural variants can often be correctly resolved. It is therefore anticipated that genome sequencing projects will routinely adopt long-read-based sequencing technologies in the coming years to gain insight in these complex genomic regions.

Yeast is a leading model organism with great importance in both basic biomedical research and biotechnological applications. Its small genome size makes it particularly suitable for long-read-based high-coverage genome sequencing. The resulting complete genome assembly with fully-resolved subtelomere structure can in turn illuminate the genetic basis of many complex phenotypic traits with unprecedented resolution. Recently, we used the long-read sequencing technologies to generate the first panel of population-level end-to-end reference genomes of 12 yeast strains representing major subpopulations of the partially domesticated *S. cerevisiae* and its sister species *Saccharomyces* *paradoxus*7. In addition, there have been a number of other studies carrying out long-read sequencing for many *S. cerevisiae* strains8–11. Given the vast genomic and phenotypic diversity of *S. cerevisiae*, we expect the incoming collection of long-read-based high-quality genome assemblies of strains from widespread geographic locations and ecological niches will substantially deepen our understanding in the *S. cerevisiae* natural genetic variation and its associated biotechnological values.

### Overview of the LRSDAY workflow

Here we present a highly organized and modular computational framework named Long-Read Sequencing Data Analysis for Yeasts (LRSDAY), which enables automated high-quality yeast genome assembly and annotation production from raw long-read sequencing data. The prototype of LRSDAY has been developed to generate the Yeast Population Reference Panel (YPRP) (<https://yjx1217.github.io/Yeast_PacBio_2016/welcome/>) in our previous study7. Under the hood, LRSDAY contains a series of task-specific modules handling long-read-based *de novo* genome assembly, long-read and short-read based assembly polishing, reference-guided assembly scaffolding, as well as comprehensive genomic feature annotations. These tasks can be run individually, selectively or coordinately depending on users’ needs. LRSDAY supports both leading long-read sequencing technologies: PacBio and Oxford Nanopore. Running the full LRSDAY workflow, the final output is a chromosome-level genome assembly with high-quality annotations of centromeres, mitochondrion, protein-coding genes, tRNAs, transposable elements (TEs; TY1-TY5 for *S. cerevisiae* and *S. paradoxus* as well as TSU4 that was likely horizontally transferred from *S. uvarum*12), and telomere associated core X and Y’ elements. LRSDAY is shipped with various auxiliary scripts, configuration files and supporting data that enable its semi-automatic installation, configuration, and execution with minimal manual intervention (Appendix 1). This design concept greatly alleviates the technical barrier for bench biologists with limited bioinformatics experiences. In addition, a real case example and its final outputs are also provided for users’ test and comparison. All these task-specific modules, auxiliary files, installed tools, sample outputs, together with the user-created project directories for the testing example and users’ own data are hosted under the same home directory ($LRSDAY\_HOME) in a self-contained fashion (Figure 1). This design makes LRSDAY well-isolated from the rest of the system and therefore greatly improves its portability. To sum up, LRSDAY is a highly transparent, automated and powerful computational framework that handles both genome assembly and annotation, which suits the needs of the yeast community in performing long-read-based genome sequencing projects. In the PROCEDURE section of this article, we provide a step-by-step walkthrough on how to install, configure, and run LRSDAY with our prepared testing example.

Table

Description automatically generated

***Figure 1. Overview of the LRSDAY directory system****. All the top-level directories (boxes, solid lines) and individual files of LRSDAY are listed and briefly described. Additional directories and files will be generated during the installation or execution of LRSDAY (boxes, dashed lines).*

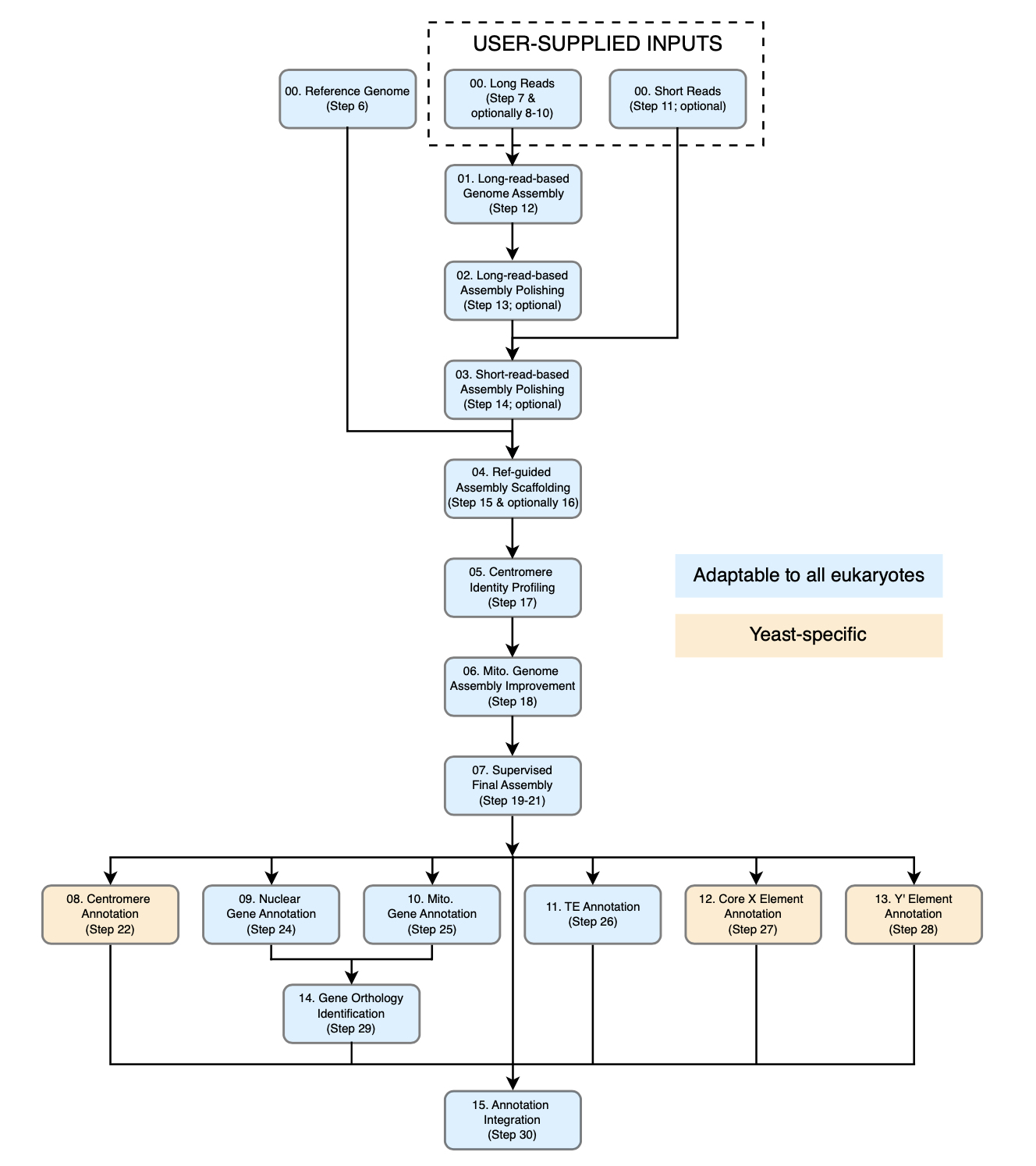
### Comparison with other methods

Genome assembly is a rapidly moving field, co-evolving with the fast-paced development of sequencing technologies. In recent years, both hybrid (i.e. using both long and short reads) and native (i.e. using long reads only) assemblers that support long-read sequencing data have been developed and tested on many different organisms8,13–18. As for gene annotation, there is also a wide range of choices that perform gene model prediction in an *ab initio* fashion or based on additional evidences (e.g. mRNA transcripts, protein-sequence alignment)19–21. Specifically for yeasts, a web-based gene annotation tool has been developed that combines both approaches22. However, there is currently no integrated solution that handles both genome assembly and annotation in a seamless way. To fill this gap, we revamped our original workflow for deriving the yeast population-level reference genomes7 into a self-contained package to considerably streamline this process with modular design and automated implementation. Moreover, rather than simply combining the existing tools for genome assembly and gene annotation, LRSDAY assembled a well-integrated workflow with many other functionalities (e.g., reference-guided scaffolding, gene orthology identification, and additional genomic feature annotation) built in, which makes it a unique one-stop solution for high-quality genome assembly and annotation production from long-read sequencing data.

### Experimental Design

Genome assembly and annotation are complex computational processes with many intermediate steps and inputs/outputs involved. With LRSDAY, we designed a highly structured project directory system to help users to run the whole workflow in an organized and modular way (Figure 2). Within such project directory system, the three subdirectories holding the pre-shipped reference genome as well as the user-supplied long (PacBio or Oxford Nanopore) and short (Illumina) reads are numbered as “00” and the task-specific subdirectories are numbered sequentially from “01” to “15” according to their execution orders. For each subdirectory, a self-explained name is attached after the number index to help users to navigate through the workflow. To run each task, users only need to edit (i.e., to specify the input and output file paths and certain task-specific parameters) and execute the task-specific pipeline scripts pre-placed in these subdirectories. These pipeline scripts will automatically set environment variables, process the data, and formulate the results. All computationally intensive tasks can be processed using multiple threads to substantially save computation time. Although LRSDAY is mainly designed for yeasts, most of these tasks can be further adapted for analysis on any eukaryotic organisms (See Appendix 2 for details). Below we briefly describe the computational processes executed by each task-specific module in LRSDAY with the corresponding PROCEDURE Step labeled in parentheses.

1. **Long\_Reads** (Step 7 & optionally 8-10): LRSDAY provides several scripts to perform filtering, downsampling, and format conversion for long reads generated from PacBio or Oxford Nanopore technologies. The read filtering script leverages between read length and read quality. With the same script, reads involved with the Nanopore adapter contamination (which is likely to happen) will also be detected and removed. The read format conversion script can convert the PacBio RSII reads in the bax.h5 format to the newer bam format, making them compatible with downstream analysis such as long-read-based assembly polishing. In addition, starting with v1.5.0, we provided a dedicated script for the basecalling, demultiplexing, and read profile plotting of raw Nanopore reads.
2. **Long-read-based\_Genome\_Assembly** (Step 12): Long reads generated from PacBio or Oxford Nanopore technologies are used to perform *de novo* genome assembly. Several assembly tools and strategies are supported to provide the flexibility to handle different use cases.
3. **Long-read-based\_Assembly\_Polishing** (Step 13; optional): Starting from the version v1.2.0, LRSDAY can perform signal-level assembly polishing directly using the long reads in their native formats (\*.bax.h5 for PacBio RSII reads, \*.bam for PacBio Sequel reads and \*.fast5 for Oxford Nanopore reads) generated from the sequencing machine. Using different software combinations, LRSDAY aligns the PacBio and Nanopore reads to the raw long-read-based genome assembly and makes necessary correction directly based on the sequencing signal embedded in the long reads.
4. **Short-read-based\_Assembly\_Polishing** (Step 14; optional): When available, LRSDAY can make additional assembly polishing using Illumina-reads to further boost the assembly quality. In this case, trimmed Illumina reads are mapped to the long-read-based genome assembly (with or without long-read-based polishing). After some further processing (e.g., alignment sorting, mate information and read group fixing, duplicates removal, local realignment), the resulting short-read alignment file is used for directing base-level corrections (i.e., for SNP and small INDELs) for the input assembly.
5. **Reference-guided\_Assembly\_Scaffolding** (Step 15 & optionally 16): The contigs from the polished genome assembly are first aligned to the reference genome to identify their shared sequence homology, based on which reference-guided assembly scaffolding is subsequently performed. The chromosomal identity of each scaffold is labeled accordingly. Structural rearrangements captured in the contigs will remain untouched during the scaffolding.
6. **Centromere\_Identity\_Profiling** (Step 17): The pre-shipped *S. cerevisiae* centromere sequences are searched against the scaffolded assembly for chromosome-specific centromere identity profiling.
7. **Mitochondrial\_Genome\_Assembly\_Improvement** (Step 18): The polished contigs corresponding to the mitochondrial genome are re-collected from the scaffolded assembly. The mitochondrial contigs spanning over the designated starting point (the *ATP6* gene by default) are broken into subsegments to prevent assembly problems caused by the circular organization of the mitochondrial genome. The resulting contigs are then re-assembled into a single linear sequence, which is further circularized by the designated starting point. The nuclear scaffolds and the circularized mitochondrial sequence together form the improved genome assembly.
8. **Supervised\_Final\_Assembly** (Steps 19-21): A modification list containing the ordering, orientation, and naming information of each sequence from the improved genome assembly is generated for users to review and to make manual adjustment when needed. The final genome assembly is further generated based on the user-edited modification list.
9. **Centromere\_Annotation** (Step 22): The pre-shipped *S. cerevisiae* centromere sequences are searched against the final genome assembly for centromere annotation.
10. **Nuclear\_Gene\_Annotation** (Step 24): *De novo* protein-coding and tRNA gene annotations are performed for the final nuclear genome assembly, which are further leveraged by the mRNA transcripts and protein sequences alignments.
11. **Mitochondrial\_Gene\_Annotation** (Step 25):Starting from the version v1.2.0, LRSDAY added a new module to perform dedicated protein-coding gene and non-coding RNA annotation for the mitochondrial genome assembly.
12. **TE\_Annotation** (Step 26): The pre-shipped curated TE library (containing the long terminal repeats (LTRs) and internal sequences of TY1-TY5 and TSU4 by default) is searched against the final genome assembly to identify TEs. The identified TEs are further curated and classified into the full-length, truncated, and solo-LTRs of TY1-TY5 and TSU4.
13. **Core\_X\_Element\_Annotation** (Step 27): The pre-shipped curated hidden Markov model (HMM) of the *S. cerevisiae* core X elements is searched against the final genome assembly to annotate core X elements.
14. **Y\_Prime\_Element\_Annotation** (Step 28): The pre-shipped representative *S. cerevisiae* Y’ element sequence is searched against the final genome assembly to annotate Y’ elements. Note that Y’ elements can have long, short or degenerated forms23, and we used a representative long-form Y’ element as the query to maximize detection power.
15. **Gene\_Orthology\_Identification** (Step 29): The annotated nuclear and mitochondrial protein-coding genes are compared with the reference protein-coding genes based on both sequence homology and gene order conservation to identify gene orthology relationship between these two sets. Based on such gene orthology relationships, the *Saccharomyces* Genome Database (SGD; <http://www.yeastgenome.org/>) systematic names are assigned to the annotated protein-coding genes.
16. **Annotation\_Integration** (Step 30): The annotations of centromeres, TEs, protein-coding genes, tRNAs, as well as core X and Y’ elements are combined and sorted to form a final integrated multi-feature annotation.



***Figure 2. The workflow of LRSDAY.*** *Each box represents an individual module. These modules are numbered according to the tasks described in Experimental Design, with the corresponding protocol step numbers also indicated. Modules that can be adapted for other eukaryotes are colored in light blue while those yeast-specific are colored in orange.*

### ANTICIPATED RESULTS

Upon the completion of the LRSDAY workflow described above, users can expect to obtain a chromosome-level genome assembly with comprehensive genomic feature annotation, which will lay a solid foundation for all kinds of downstream genomic and functional analyses. The final genome assembly is highly continuous, with each chromosome assembled in an end-to-end fashion. Both genome-wide dotplots and summary statistics will be generated to help users to evaluate the genome assembly quality both graphically and quantitatively. As for the annotation, LRSDAY profiles a full spectrum of genomic features for the assembled yeast genome, which include centromeres, protein-coding genes, tRNAs, TY1-TY5 & TSU4 transposable elements, as well as the telomere-associated core X and Y’ elements. The availability of such rich information can be very valuable for users working on diverse biological questions.

### Limitations and potential adaptation

In its distributed form, LRSDAY is tailored for the model budding yeast *S. cerevisiae* and its closely related sister species *S. paradoxus* witha number of pre-shipped auxiliary data files configured accordingly. However, given its modular design, the backbone of LRSDAY can be adapted for virtually any eukaryotic organisms to perform genome assembly and polishing, reference-guided scaffolding, protein-coding genes and tRNA annotations, gene orthology identification, and annotation integration. In Appendix 2, we provide some tips with regard to such adaptation. Moreover, those assembly polishing, scaffolding and various annotation modules (See Experimental Design) can also be used to analyze existing genome assemblies derived from any or any combination of sequencing technologies. Such flexibility makes LRSDAY very useful for expanded use cases and therefore suits the needs of a broader audience.

### Expected improvements

As thousands of yeast strains have been or are currently under sequencing24–27, our knowledge of the overall genome content diversity28 of this important model organism is expanding rapidly, revealing a whole new picture of the pan-genome diversity of *S. cerevisiae*. For example, our lab is currently working on characterizing the pan-genome of >1,000 *S. cerevisiae* isolates across the globe (The 1002 Yeast Genomes Project27; <http://1002genomes.u-strasbg.fr/>). Future developments of LRSDAY will incorporate such pan-genome dataset to provide additional annotation information for those non-reference genes, especially with regard to their evolutionary origin, population prevalence, and putative functions. Such information will greatly help users to dissect and interpret complex genotype-phenotype interactions in diverse ecological and biotechnological settings. An additional potential future direction of our research is the direct integration with the downstream synteny analysis tools (e.g. CHROnicle29, MCScanX30, etc) to perform automatic large-scale structural variants discovery, which exploits one of the major benefits of having a high-quality genome assembly derived from long reads. Finally, we envision developing a dedicated web-based tool to implement such database and tool integration at a larger scale towards fully automated genomics analysis for the yeast community in the long run.

# CITATIONS

Jia-Xing Yue & Gianni Liti. (2018) Long-read sequencing data analysis for yeasts. *Nature Protocols*, 13:1213–1231.

Jia-Xing Yue, Jing Li, Louise Aigrain, Johan Hallin, Karl Persson, Karen Oliver, Anders Bergström, Paul Coupland, Jonas Warringer, Marco Cosentino Lagomarsino, Gilles Fischer, Richard Durbin, Gianni Liti. (2017) Contrasting evolutionary genome dynamics between domesticated and wild yeasts. *Nature Genetics*, 49:913-924.

# MATERIALS

### Hardware, operating system and network

This protocol is designed for a desktop or computing server running an x86-64-bit Linux operating system. Multi-threaded processors are preferred to speed up the process since many steps can be configured to use multiple threads in parallel. For assembling and analyzing the budding yeast genomes (genome size = ~12.5 Mb), at least 16 Gb of RAM and 100 Gb of free disk space are recommended. More disk space is needed if you want to use LRSDAY to do Nanopore basecalling. When adapted for other eukaryotic organisms with larger genome sizes, the RAM and disk space consumption will scale up, majorly during *de novo* genome assembly (performed by Canu18 by default). Please refer to Canu’s manual (<http://canu.readthedocs.io/en/latest/>) for suggested RAM and disk space consumption for assembling large genomes. Stable Internet connection is required for the installation and configuration of LRSDAY as well as for retrieving the testing data.

### Software or library requirements

* bash (<https://www.gnu.org/software/bash/>)
* bzip2 and libbz2-dev (<http://www.bzip.org/>)
* cmake (<https://cmake.org/>)
* gcc and g++ v4.9.1 or newer (<https://gcc.gnu.org/>)
* ghostscript ([https://www.ghostscript.com](https://www.ghostscript.com/))
* git (<https://git-scm.com/>)
* gnu make (<https://www.gnu.org/software/make/>)
* gzip (<https://www.gnu.org/software/gzip/>)
* java runtime environment (JRE) v1.8.0 or newer (<https://www.java.com>)
* perl v5.12 or newer (<https://www.perl.org/>)
* python v2.7.9 or newer (<https://www.python.org/>)
* python-devel
* python v3.7 or newer (<https://www.python.org/>)
* python3-devel
* tar (<https://www.gnu.org/software/tar/>)
* unzip (<http://infozip.sourceforge.net/UnZip.html>)
* virtualenv v15.1.0 or newer (<https://virtualenv.pypa.io>)
* wget v1.14 or newer (<https://www.gnu.org/software/wget/>)

●    zlib and zlib-devel (<https://zlib.net/>)

### Input data

* Long reads: A single FASTQ file containing PacBio or Oxford Nanopore reads is needed, which will be used for long-read based *de novo* genome assembly (Task 01). Optionally, long reads in their native sequencing-machine formats (\*.bax.h5 with the associated \*.metadata.xml for PacBio RSII reads, \*.bam/\*.fastq for PacBio Sequel/HiFi reads and \*.fast5 for Oxford Nanopore reads) are needed for performing long-read-based assembly polishing (Task 02).
* Short reads: Short reads are optional for LRSDAY but LRSDAY could take advantage of short reads when such data is available to perform additional assembly polishing (Task 03). If paired-end Illumina sequencing is performed, two FASTQ files containing the forward and reverse Illumina reads respectively are needed. If only single-end Illumina sequencing data is available, one FASTQ file containing the single-end reads is needed.
* Reference genome: For the budding yeast *S. cerevisiae*, we pre-shipped two reference genome files (one original assembly and one with hard-masked subtelomeres and chromosome-ends based on our previous study7). The masked version is used for chromosomal scaffolding to minimize the confounding effect due to interchromosomal subtelomeric rearrangements. When working with organisms of which the subtelomeric regions are undefined, users can just use a single raw reference genome instead. The reference genome file(s) will be used for reference-guided scaffolding, mitochondrial genome assembly improvement, and supervised final genome assembly (Task 04, 06 and 07 respectively).
* A number of *S. cerevisiae*-specific auxiliary data have been pre-shipped with LRSDAY for genomic feature annotation and gene orthology identification (Task 05 and 08-14).

### Example data

* The *S. cerevisiae* reference genome pre-shipped with LRSDAY is taken from our previous study7 with the Genbank accession number GCA\_002057635.1 (<https://www.ncbi.nlm.nih.gov/assembly/GCA_002057635.1/>). The sequencing reads used for the testing example come from our recent study31 (ENA bioproject accession: PRJEB50706), which consists of both Oxford Nanopore and Illumina reads produced from the *S. cerevisiae* strain CPG\_1a. The Oxford Nanopore reads can be retrieved with the ENA URL (<https://ftp.sra.ebi.ac.uk/vol1/run/ERR853/ERR8531810/CPG_1a.tar.gz> ). The Illumina reads can be retrieved with the ENA URL (<https://ftp.sra.ebi.ac.uk/vol1/run/ERR860/ERR8603909/CPG_1a_1.fq.gz> & <https://ftp.sra.ebi.ac.uk/vol1/run/ERR860/ERR8603909/CPG_1a_2.fq.gz> )

# PROCEDURE

### *Download, install and configure LRSDAY*

1. Download the latest LRSDAY release (current version: v1.7.1) by entering the following commands in a terminal window:

$ wget <https://github.com/yjx1217/LRSDAY/releases/download/v1.7.1/LRSDAY-v1.7.1.tar.gz>   
$ tar xvzf LRSDAY-v1.7.1.tar.gz

$ cd LRSDAY-v1.7.1

$ bash install\_dependencies.sh

**[Important Note]**

* Please note that it will take quite a while for the installation to finish. Therefore, it is recommended to run the bash script above with nohup (<https://en.wikipedia.org/wiki/Nohup>), which prevents the unintended interruption of the running script. The same trick applies to all the other module-specific bash script as well. For this installation, the nohup version of the command could be:  
  $ nohup bash ./install\_dependencies.sh > run.log.txt 2>&1 &
* Some dependencies (e.g., pb-assembly, bax2bam) are only kept for supporting historical PacBio reads, and it is quite time-consuming to install them. Therefore, for most users, it is highly recommended to install LRSDAY in the lite mode by specifying the “-l” option, which will skip the installation of these dependencies. An example of such lite installation is as follows:

$ nohup bash ./install\_dependencies.sh -l > run.log.txt 2>&1 &

If later you found that you still want these dependencies, you can easily patch the existing lite installation by running the following command, which will skip all existing installations and only install those extra dependencies:

$ nohup bash ./install\_dependencies.sh > run.log.txt 2>&1 &

Please note if the installation script prompts for the following message at the end of the installation process:

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Your java version is not the version required by LRSDAY (java v1.8)!

Please manually set the directory path to java 1.8 executable on the last line of the env.sh file generated by this installation script!"

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

If this message is prompted, please manually modify the last line of the env.sh file to provide the path to the java 1.8 executable accordingly after the installation process successfully finishes.

If the installation succeeds, you should see the following massage:

“LRSDAY message: This bash script has been successfully processed! :)”

This signifies the success of the installation process. The same is true for all module-specific bash scripts (named as “LRSDAY.\*.sh”) of LRSDAY.

Upon the success of the installation, a subdirectory named build and a file named env.sh will be generated. The build subdirectory holds all the installed dependencies, while the env.sh file contains the execution paths of these dependencies. This file will be automatically loaded to set up the working environment for LRSDAY’s various modules. The base directory of LRSDAY is defined as $LRSDAY\_HOME in this file.

In case of installation failure (most likely due to internet connection problem that might occur temporarily), the users only need to re-run the installation script install\_dependencies.sh. LRSDAY will automatically detect the previous interruption point and resume the installation process.

**Major outputs when running this step:**

* build # The subdirectory holding all the installed dependencies.
* env.sh # The file containing the execution paths of these dependencies.

**[Important Note]**

Although we made the effort to cover as much as technical details in our pre-shipped the installation script. Unexpected installation errors might still be encountered by some users due to server-specific problems. In that case, it is possible to skip the installation of certain problematic tool with LRSDAY while using a separately installed copy instead. To do this, just mask out (by putting “#” at the beginning of the corresponding lines) the installation commands in the install\_dependencies.sh script. In this case, by running the install\_dependencies.sh script, LRSDAY will still install other third-party dependencies and generated the env.sh file as usual. The only difference is that in the resulting env.sh file, the environmental variable to the skipped tool will have blank value. User just need to manually add the accessible path (the full directory to the corresponding executable) to a separately installed copy of the skipped tool into the env.sh file, so that LRSDAY can correctly locate and summon the separately installed tool by loading env.sh file. This should work in most cases. If users still have installation problems, please do not hesitate to report the issue via GitHub’s issue ticket system (<https://github.com/yjx1217/LRSDAY/issues>), so that we can provide one-to-one help.

1. Load the environment settings for LRSDAY by entering:

$ source env.sh

After loading the pre-configured environment settings, the current directory should be assigned to the environment variable $LRSDAY\_HOME. You can check to see if the full path to your current directory is displayed after entering:

$ echo $LRSDAY\_HOME

**[Important Note]**

While almost all required tools have been automatically installed and configured, manual configuration is needed for RepeatMasker32. Make sure to run this command to load the pre-configured environment settings before the manual setup described in Step 3-4. If you exited your terminal session before or in the middle of such manual setup, you need to re-load the environment settings before proceeding. These environment settings will be automatically loaded each time the task-specific bash pipelines of LRSDAY are executed.

1. Obtain the installation paths of TRF and rmblastn/makeblastdb by entering:

$ echo $trf\_dir

$ echo $rmblast\_dir

Remember these two paths since they will be used for the RepeatMasker configuration in Step 4.

1. Run the configuration script for RepeatMasker by entering:

$ cd $repeatmasker\_dir

$ perl ./configure

This configuration script will prompt for several questions. Please do the following to answer these questions. Enter “env” for the question about the installation path of Perl. Just press enter for the question about the installation path of RepeatMasker. Enter the first path that you obtained in Step 3 for the question about the installation path of TRF. Enter “2” for the question about selecting a search engine. Then enter the second path that you obtained in Step 3 for the question about the installation path of rmblastn and makeblastdb. Just press enter for the question about the default search engine. And finally enter “5” to complete the configuration.

### *Run LRSDAY with the testing example*

1. Create the project directory. When running LRSDAY with your own data, it is recommended to make a copy of our Project\_Template directory to create your own project directory such as Project\_abc, where “abc” can be any string containing letters, numbers, or underscores. For this testing example, we make a copy of the Project\_Template directory and name it as Project\_Example by entering:

$ cd $LRSDAY\_HOME

$ cp -r Project\_Template Project\_Example

Before proceeding to your own project, it is advised to first run our prepared testing example to check if LRSDAY is working properly as well as to get acquainted with the logic and workflow of LRSDAY.

1. Prepare the reference genome files. When running LRSDAY with your own data, you can directly put the reference genome (in FASTA format without compression) in the 00.Reference\_Genome subdirectory of your project directory (e.g. Project\_abc). If your sequenced organism is *S. cerevisiae* or *S. paradoxus*, you can use the reference genome pre-shipped with LRSDAY. Here we prepare the pre-shipped reference genome for the testing example by entering:

$ cd ./Project\_Example/00.Reference\_Genome

$ bash LRSDAY.00.Prepare\_Sc\_Reference\_Genome.sh

**Major outputs when running this step for the testing example:**

* S288C.ASM205763v1.fa # The preprocessed reference genome file in FASTA format.
* S288C.ASM205763v1.noncore\_masked.fa # The preprocessed reference genome file in FASTA format with its subtelomeric and chromosome-end region hard masked.

1. Prepare the raw Oxford Nanopore reads. When running LRSDAY with your own data, you can directly put the long reads in the 00.Long\_Reads subdirectory of your project directory (e.g. Project\_abc). The FASTQ reads file should be placed directly in this directory and compressed files with file extensions of “.gz” are supported. When available, the sequencing reads in their sequencing platform’s native formats should be placed in the directory pacbio\_fofn\_files (\*.bax.h5 and the associated \*.metadata.xml for PacBio RSII reads or \*.bam file for PacBio Sequel reads along with a fofn file that contains their absolute paths) or the directory nanopore\_fast5\_files (\*.fast5 for Oxford Nanopore reads). Starting with v1.7.0, LRSDAY provides a dedicated bash script (LRSDAY.00.Nanopore\_Reads\_Basecalling.sh) for performing basecalling, demultiplexing, and quality control summary of the basecalled/demultiplexed Nanopore reads. Starting with v1.6.0, LRSDAY further provides a bash script (“LRSDAY.00.Summary\_Report\_for\_Long\_Reads.sh”) for generating and plotting summary statistics for the fastq(.gz) formatted long reads. For this testing example, first navigate to the provided 00.Long\_Reads/nanopore\_raw\_fast5\_files directory to download the raw Oxford Nanopore raw reads (in fast5 format) and uncompress it:

$ cd ./../00.Long\_Reads/nanopore\_raw\_fast5\_files

$ wget <https://ftp.sra.ebi.ac.uk/vol1/run/ERR853/ERR8531810/CPG_1a.tar.gz>

$ tar xvzf CPG\_1a.tar.gz

1. (Optional) Run basecalling for the raw nanopore fast5 reads. This step is only needed if your input long reads are Oxford Nanopore raw fast5 reads, which is the case for our testing example.

Edit the LRSDAY.00.Nanopore\_Reads\_Basecalling.sh script to specify the basecalling parameters. There are many parameters in this file but the ones that you need to adjust are only the following:

* gpu\_run\_mode # The default value “cpu” will work fine for all projects. But if your computing server has GPU and CUDA (<https://developer.nvidia.com/cuda-zone>) support. It is highly recommended to set this to “gpu” to substantially speed up the basecalling process. And in this case, you will also need to adjust the default values for gpu\_bin\_path, gpu\_lib\_path, and gpu\_include\_path based on your own CUDA environment.
* sample\_id
* flowcell\_version
* sequencing\_kit\_version

For our testing example, Figure 3 shows how the basecalling parameters should be set for the sample CPG\_1a (Figure 3).

Text

Description automatically generated

***Figure 3. The customizable parameter setting section in the LRSDAY.00.Nanopore\_Reads\_Basecalling.sh bash script.***

Once this is all set. Run the bash script to perform basecalling and quality-control summarization.

$ cd 00.Long\_Reads

$ bash LRSDAY.00.Nanopore\_Reads\_Basecalling.sh

Upon finishing, the basecalled reads in fast5 format will be placed under:  
Project\_Example/00.Long\_Reads/nanopore\_basecalled\_fast5\_files/<sample\_id>

The basecalled reads in fastq format will be place under:

Project\_Example/00.Long\_Reads/nanopore\_basecalled\_fast5\_files/<sample\_id>

The quality control summary report for the basecalled reads will be place under:

Project\_Example/00.Long\_Reads/nanopore\_basecalled\_summary\_files/<sample\_id>

**TABLE 1** | Basecalling summary statistics of the *S. cerevisiae* strain CPG\_1a raw Oxford Nanopore reads for the testing example.

|  |  |
| --- | --- |
| **Summary statistics** | **Values** |
| Mean read length | 14,603.50 bp |
| Mean read quality | 12.70 |
| Median read length | 6,480.00 bp |
| Median read quality | 13.20 |
| Number of reads | 170,167.00 |
| Read length N50 | 33,344.00 |
| STDEV read length | 20,199.50 |
| Total bases | 2,485,027,602 bp |

Chart, scatter chart

Description automatically generated

***Figure 4. The distribution of read quality and read length after basecalling.***

1. (Optional) Edit and run the provided bash script LRSDAY.00.Long\_Reads\_Preprocessing.sh to perform read filtering/downsampling as well as adaptor removal (for Nanopore reads only) for your raw reads. This step is highly recommended for Nanopore reads regarding adaptor removal. For this testing example, run the following command to downsampling our input reads to 80X to reduce the computational consumption without much compromise in assembly quality:

$ bash LRSDAY.00.Long\_Reads\_Preprocessing.sh

**Major outputs when running this step for the testing example:**

* CPG\_1a.porechop.fastq.gz # The adapter-chopped long reads.
* CPG\_1a.porechop.summary.txt # The summary report for adapter chopping.
* CPG\_1a.filtlong.fastq.gz # The filtered long reads. This file will be used for long-read-based genome assembly.

The downsampling implemented here leverages both quality and length of the input reads. Alternatively, you can also perform downsampling using our pre-shipped Perl script subsampling\_sequences.pl (in the $LRSDAY\_HOME/scripts directory):

$ perl $LRSDAY\_HOME/scripts/subsampling\_sequences.pl -i input.fq(.gz) -f fastq -s 0.1 -m random -p output

# randomly sampling the 10% sequences

or

$ perl $LRSDAY\_HOME/scripts/subsampling\_sequences.pl -i input.fq(.gz) -f fastq -s 0.1 -m longest -p output   
# sampling the 10% longest sequences

**[Important Note]**

We recommend running this step and all the other time-consuming steps with “nohup”, which allows the process to continue running after you exit the terminal or logout from the server. As an example, you can run the bash script using nohup as follows:

$ nohup bash LRSDAY.00.Long\_Reads\_Preprocessing.sh >run\_log.txt 2>&1 &

1. (Optional) If your long reads are generated from the PacBio Sequel platform, your reads are likely to be in BAM format. In this case, convert it to FASTA or FASTQ format for genome assembly using the following commands:

$ source ./../../env.sh

$ $bedtools\_dir/bedtools bamtofastq -i long\_reads.bam -fq long\_reads.fastq

$ gzip long\_reads.fastq

1. (Optional) Prepare short reads. Short reads are not required by LRSDAY but it is good to have for additional assembly polishing. When running LRSDAY for your own data, put your short reads (i.e. Illumina reads) in the 00.Short\_Reads subdirectory of your project directory (e.g. Project\_abc). The reads file should be in FASTQ format with “gzip” comprehension (identified by the “.gz” extension). For this testing example, you can download the Illumina reads by entering:

$ cd ./../00.Short\_Reads

$ wget <https://ftp.sra.ebi.ac.uk/vol1/run/ERR860/ERR8603909/CPG_1a_1.fq.gz>

$ wget <https://ftp.sra.ebi.ac.uk/vol1/run/ERR860/ERR8603909/CPG_1a_2.fq.gz>

After downloading, we recommend to rename the short reads file name as <sample\_id>.R1.fastq.gz and <sample\_id>.R2.fastq.gz for downstream script to automatically pick up the correct read paths without further specification. For the testing example, you can do this by entering:

$ mv CPG\_1a\_1.fq.gz CPG\_1a.R1.fastq.gz

$ mv CPG\_1a\_1.fq.gz CPG\_1a.R1.fastq.gz

1. Perform long-read-based *de novo* genome assembly. The use of multi-threading and nohup is highly recommended for this step. Starting genome assembly by running the following commands:

$ cd ./../01. Long-read-based\_Genome\_Assembly

$ bash LRSDAY.01.Long-read-based\_Genome\_Assembly.sh

Upon the completion of this step, a summary file (CPG\_1a.assembly.raw.stats.txt for this testing example) will be generated to report some basic summary statistics (e.g. total assembly size, N50 (i.e. the contig length such that 50% of the total assembly size is contained in contigs of at least this size), L50 (i.e. the number of longest contigs such that 50% of the total assembly size is contained), GC-content, etc) to assist gauging the genome assembly quality (Table 1). Two VCF files (CPG\_1a.assembly.raw.filter.mummer2vcf.SNP.vcf and CPG\_1a.assembly.raw.filter.mummer2vcf.INDEL.vcf for the testing example) will also be generated to report base-level differences between the raw genome assembly and the reference genome for their uniquely alignable regions, which could also help for assessing the genome assembly quality.

**[Important Note]**

* Starting from v1.1.0, LRSDAY added the "customized\_canu\_parameters” option to support customized parameter settings for the Canu assembler. In addition, additional assemblers such as Flye, wtdbg2, smartdenovo, and shasta are further supported. Based on our test, these additional assemblers all ran significantly faster than Canu but usually also came with understandable tradeoff in assembly precision, as reflected by the higher base-level error rate of their resulting assemblies. Therefore, when running genome assembly with these alternative assemblers, post-assembly polishing is strongly recommended. Alternatively, you can also run this step using two assemblers together by set assember=“canu-flye” or “canu-wtdbg2” or “canu-smartdenovo” or “canu-shasta” ), in which LRSDAY will let Canu to generate self-corrected long reads and then assemble the genome with the selected alternative assemblers (i.e. Flye, wtdbg2, smartdenovo, or shasta) based on corrected long reads. Starting from v1.6.0, LRSDAY natively supports running Canu in the TrioBinning mode for phased diploid assembly.
* When running LRSDAY with your own data, modify the bash script to specify the input reads, the reference genome, the input reads type (e.g. “pacbio\_raw”, “pacbio\_corrected”, “pacbio\_hifi”, “nanopore\_raw” or “nanopore\_corrected”), the estimated genome size for the assembled genome, the assembly strategy (i.e. different assemblers or assembler combinations) and parameters (for Canu only) that you are going to use, as well as the prefix for the output data. Remember to do similar project-specific adjustment for all the following steps.

Chart, line chart

Description automatically generated

***Figure 5. The dotplot comparison between the CPG\_1a raw assembly and the S. cerevisiae reference genome.***

**Major outputs when running this step for the testing example:**

* CPG\_1a.assembly.raw.fa # The long-read-based de novo genome assembly containing all the contigs assembled by Canu. This file is soft-linked to the file CPG\_1a.assembly.canu.fa
* CPG\_1a.assembly.raw.stats.txt # The summary table reporting basic assembly statistics, such as the number of the assembled sequences, the total length of the assembled sequences, the minimal, maximal, mean and median lengths of the assembled sequences, the N50, L50, N90, and L90 of the assembled sequences, as well as the base composition (A%, T%, G%, C%, AT%, GC% and N%) of the assembled sequences.
* CPG\_1a.assembly.raw.filter.pdf # The genome-wide dotplot for the comparison between the raw genome assembly and the reference genome.
* CPG\_1a.assembly.raw.filter.mummer2vcf.SNP.vcf # The VCF file showing SNP differences between the raw genome assembly and the reference genome.
* CPG\_1a.assembly.raw.filter.mummer2vcf.INDEL.vcf # The VCF file showing INDEL differences between the raw genome assembly and the reference genome.

1. Polishing genome assembly with long-reads. To compensate for the comparatively lower quality of long-read sequencing data in general, assembly polishing is strongly recommended. When you have long reads in their sequencing platform’s native formats, we recommend running the first-pass polishing for the assembly generated in Step 12 based on long reads. When sequenced with PacBio HiFi technology, long-read-based polishing will be automatically skipped. When sequenced with PacBio CLR technology, this is done by using PacBio’s own quiver/arrow pipeline14. If your long reads are generated by Oxford Nanopore sequencing, we recommend using racon-medaka (works better in our tests) or nanopolish. The long-reads in native sequencing format are needed for quiver/arrow and nanopolish. To achieve the best quality, one can run this step using full reads set even when the downsampled set was used for deriving *de novo* genome assembly (Step 12). Also, starting from v1.4.0, LRSDAY enables running multi-round polishing by setting the ‘rounds\_of\_successive\_polishing=’ parameter to minimize remaining sequencing errors. This step can be run with multiple threads. Perform long-read-based assembly polishing by the following command:

$ cd ./../02.Long-read-based\_Assembly\_Polishing

$ bash LRSDAY.02.Long-read-based\_Assembly\_Polishing.sh

**Major outputs when running this step for the testing example:**

* CPG\_1a.assembly.long\_read\_polished.fa # The long-read-based polished assembly generated by three rounds of racon-medaka polishing. This file is soft-linked to the file CPG\_1a.assembly.medaka.round\_3.fa

1. (Optional) Polishing genome assembly with Illumina reads. When Illumina reads are available, we recommend running this additional polishing step either for the raw assembly generated in Step 12 (when Step 13 is skipped) or for the long-read-polished assembly generated in Step 13 (as with our testing example). Also, like for Step 13, LRSDAY can also run multi-round polishing in this step by setting the ‘rounds\_of\_successive\_polishing=’ parameter starting from v1.4.0. Use the following commands to perform Illumina-read-based assembly polishing. This step can be run with multiple threads.

$ cd ./../03.Short-read-based\_Assembly\_Polishing

$ bash LRSDAY.03.Short-read-based\_Assembly\_Polishing.sh

**Major outputs when running this step for the testing example:**

* CPG\_1a.assembly.short\_read\_polished.fa # The long-read-based polished assembly generated by three rounds of pilon polishing. This file is soft-linked to the file CPG\_1a.assembly.pilon.round\_3.fasta

**TABLE 2** | Assembly statistics for the genome of *S. cerevisiae* strain CPG\_1a assembled in the testing example.

|  |  |  |  |
| --- | --- | --- | --- |
| **Assembly statistics** | **Raw assembly** | **Scaffolded assembly** | **Final assembly** |
| Total sequence count | 22 | 22 | 17 |
| Total sequence length (bp) | 12585690 | 12565185 | 12208964 |
| Min sequence length (bp) | 8900 | 8914 | 87602 |
| Max sequence length (bp) | 1531332 | 1532830 | 1532830 |
| Mean sequence length (bp) | 572076.82 | 571144.77 | 718174.35 |
| Median sequence length (bp) | 552859.00 | 553474.00 | 739175.00 |
| N50 (bp) | 943564 | 944461 | 944461 |
| L50 | 6 | 6 | 6 |
| N90 (bp) | 337884 | 338269 | 447596 |
| L90 | 14 | 14 | 13 |
| GC% | 37.57 | 37.61 | 38.22 |
| N% | 0.00 | 0.00 | 0.00 |

Note

N50: the contig length such that 50% of the total assembly size is contained in contigs of at least this size. L50: the number of longest contigs such that 50% of the total assembly size is contained. N90: the contig length such that 90% of the total assembly size is contained in contigs of at least this size. L90: the number of longest contigs such that 90% of the total assembly size is contained. GC%: the percentage of guanine (G) and cytosine (C) bases in the nucleotide sequences. N%: the percentage of the N bases in the nucleotide sequence. In genome assembly, the N bases are usually used to represent scaffolding gaps.

1. Perform chromosome-level scaffolding for the long-read-based assembly. Two scaffolders are supported: RagTag33 and Ragout34. RagTag (by default in LRSDAY since v1.7.0) is generally recommended given its superior processing speed. For the testing example, please run the following commands:

$ cd ./../04.Reference-guided\_Assembly\_Scaffolding

$ bash LRSDAY.04.Reference-guided\_Assembly\_Scaffolding.sh

This step can be run with multiple threads. Upon completion, a list of summary statistics (CPG\_1a.assembly.ref\_based\_scaffolded.stats.txt for this testing example) will be generated for the scaffolded assembly (Table 1).

Please check the generated genome-wide dotplot (CPG\_1a.assembly.ref\_based\_scaffolded.filter.pdf for the testing example) (Figure 6) to verify the correctness of chromosomal identity assignment performed by RagTag or Ragout and apply manual adjustment in Step 20 when necessary. When running LRSDAY with your own data, you might see a single scaffold corresponds to more than one reference chromosomes, which could be due to shared sequence homology between duplicated regions or interchromosomal rearrangements. Both types of events can be correctly interpreted based on the genome-wide dotplot generated in this step. In either case, LRSDAY can correctly assign chromosomal identity of the corresponding scaffold based on its encompassed centromere identity as annotated in Step 17.

Chart

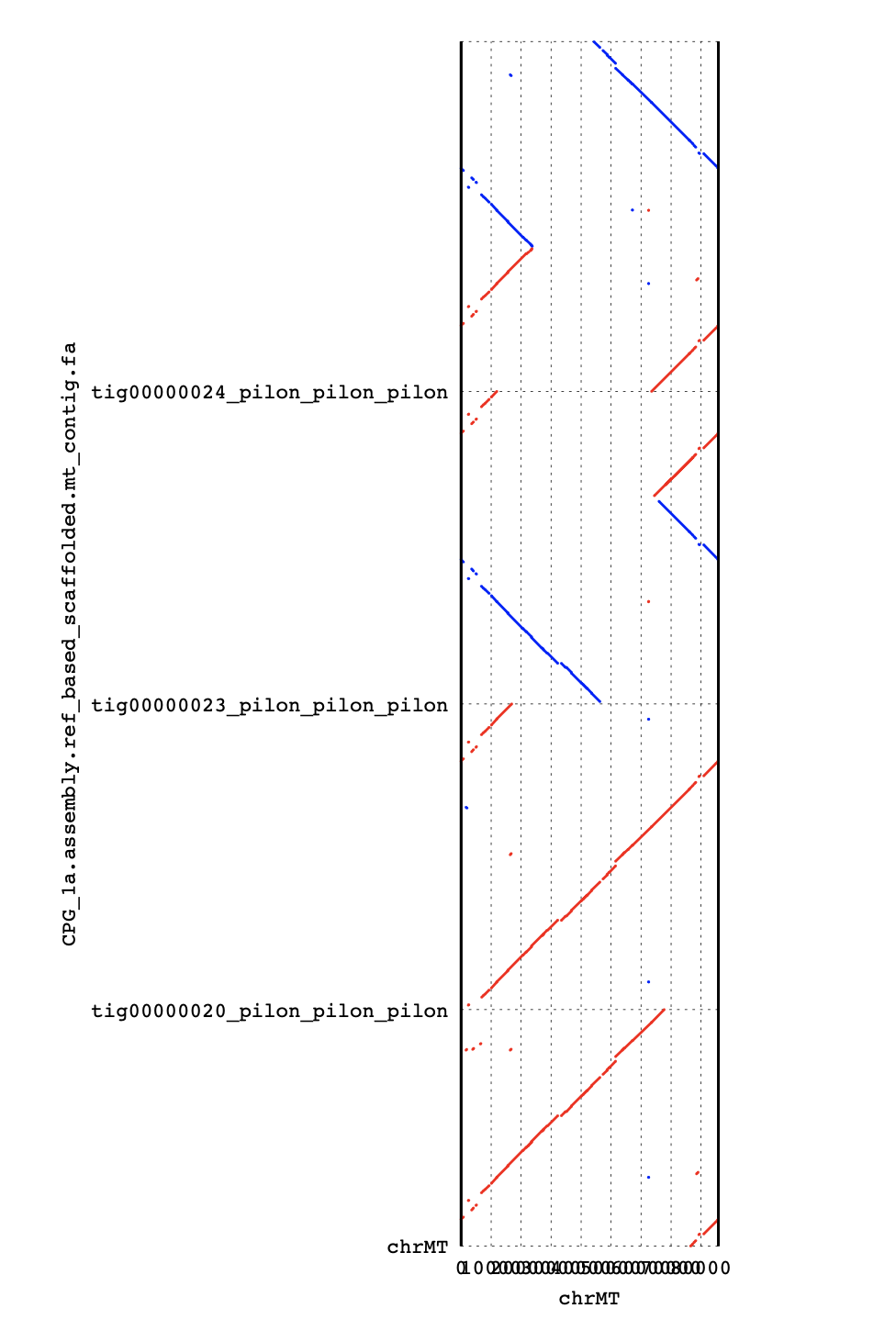
Description automatically generated

***Figure 6. The dotplot comparison between the CPG\_1a scaffolded assembly and the S. cerevisiae reference genome.***

Due to the high AT and repeat contents and the circular conformation of the mitochondrial genome, multiple contigs corresponding to the mitochondrial genome are often obtained from the raw genome assembly, as shown in the generated mitochondrial genome dotplot (CPG\_1a.assembly.ref\_based\_scaffolded.chrMT.filter.pdf for the testing example) (Figure 7). A list of such mitochondrial contigs will also be generated (CPG\_1a.assembly.ref\_based\_scaffolded.mt\_contig.list for the testing example), which will be used in Step 18 for improving mitochondrial genome assembly. You can manually edit the CPG\_1a.assembly.ref\_based\_scaffolded.mt\_contig.list file to remove or add contigs corresponding to chrMT for the downstream chrMT assembly improvement (Step 18). For our testing example, based on the dotplot, we think the apparent inversions in the contigs tig00000023\_pilon\_pilon\_pilon and tig00000024\_pilon\_pilon\_pilon should be derived from mis-assembly, which can be further checked by the long- and short- reads to assembly alignment generated by Step 13 and Step 14. Therefore, we make manual edits in the CPG\_1a.assembly.ref\_based\_scaffolded.mt\_contig.list file by removing the two lines corresponding to these two contigs. Therefore, the edited file only contains the following to contigs:

chrMT

tig00000020\_pilon\_pilon\_pilon



***Figure 7. The dotplot comparison between the CPG\_1a scaffolded mitochondrial genome (chrMT)assembly and the S. cerevisiae reference mitochondrial genome (chrMT).***

**Major outputs when running this step for the testing example:**

* CPG\_1a.assembly.ref\_based\_scaffolded.fa # The scaffolded genome assembly based on the reference genome.
* CPG\_1a.assembly.ref\_based\_scaffolded.stats.txt # The summary table reporting basic assembly statistics of the scaffolded genome assembly.
* CPG\_1a. assembly.ref\_based\_scaffolded.filter.pdf # The genome-wide dotplot for the comparison between the scaffolded assembly and the reference genome.
* CPG\_1a.assembly.ref\_based\_scaffolded.mt\_contig.list # The list of assembled contigs corresponding to the mitochondrial genome. This file will be used for Step 18.
* CPG\_1a.assembly.ref\_based\_scaffolded.mt\_contig.fa # The assembled contig sequences corresponding to the mitochondrial genome.
* CPG\_1a.assembly.ref\_based\_scaffolded.chrMT.filter.pdf # The dotplot for the comparison between the scaffolded mitochondrial genome assembly and the reference mitochondrial genome.

1. (Optional) When running LRSDAY for your own data, if you have strong evidence for mis-scaffolding based on prior knowledge or other experimental data (e.g., mate-pair libraries or chromosomal contact data), break the corresponding ragout scaffolds back to contigs and re-joined them with corrected order using the pre-shipped Perl scripts break\_scaffolds\_by\_N.pl, join\_contigs\_by\_N.pl and extract\_region\_from\_genome.pl in the $LRSDAY\_HOME/scripts directory by running the following commands:

$ perl $LRSDAY\_HOME/scripts/break\_scaffolds\_by\_N.pl -i <the input FASTA file containing the scaffold sequence(s) to break> -o < the output FASTA file containing the scaffolds after the breaking> -g <the minimal length of runs of Ns in the input scaffold(s) for breaking, e.g., 5000>

$ perl $LRSDAY\_HOME/scripts/join\_contigs\_by\_N.pl -i <the input FASTA file containing contigs for joining in a sequential order> -o <the output FASTA file containing scaffold sequence after the contig joining> –g <gap size, i.e., the number of Ns to be inserted between two joined contigs> -t <sequence name for the newly joined scaffold>

$ perl $LRSDAY\_HOME/scripts/extract\_region\_from\_genome.pl -i <the input FASTA file containing the genome assembly> -o <the output FASTA file containing the extracted query sequence> -q <specially formatted query string (sequence:start-end:strand) containing the genomic coordinates for the region to be extracted, e.g. using the query string chrI:1000-4000:+ for extracting the sequence in the region 1000-4000 bp on the + strand of chrI in the input genome assembly> -f <the length of flanking sequences to be extracted as well, e.g. 100 for 100-bp flanking region>

A scenario for such use case is when the breakpoints of structural rearrangements are also the breakpoints of the genome assembly. In this case, the reference-based scaffolding will arrange contigs according to the reference genome configuration and therefore un-do the genome rearrangement.

1. Perform centromere profiling for the scaffolded genome assembly by running the following commands:

$ cd ./../05.Centromere\_Identity\_Profiling

$ bash LRSDAY.05.Centromere\_Identity\_Profiling.sh

**[Important Note]**

The chromosome-specific centromere identities profiled here will be used as another layer of information for the final chromosomal identity assignment in Step 20. The profiled centromere identities usually agree well with the chromosomal identities labeled in Step 15, so that chrI will have the CEN1 centromere and chrII will have the CEN2 centromere, etc. Exception can occur when interchromosomal rearrangements are involved in your sequenced genome. In such case, we recommend naming those rearranged chromosomes according to their encompassed centromeres (annotated in this step) when deriving the final assembly in Steps 19-21.

**Major outputs when running this step for the testing example:**

* CPG\_1a.centromere.gff3 # The profiled centromere identities for the scaffolded genome assembly.

1. Perform mitochondrial genome assembly improvement by running the following commands:

$ cd ./../06.Mitochondrial\_Genome\_Assembly\_Improvement

$ bash LRSDAY.06.Mitochondrial\_Genome\_Assembly\_Improvement.sh

Check the generated final mitochondrial genome dotplot (CPG\_1a.assembly.mt\_improved.chrMT.filter.pdf for the testing example) and compare it with the mitochondrial genome dotplot generated in Step 15 to see how the mitochondrial genome assembly has been improved when aligning with the reference mitochondrial genome (Figure 8).

Chart, line chart

Description automatically generated

***Figure 8. The dotplot comparison between the CPG\_1a final mitochondrial genome (chrMT)assembly and the S. cerevisiae reference mitochondrial genome (chrMT).***

When running this step for your own data, the degree of such improvement may vary because it depends on both the complexity of the assembled mitochondrial genome and the quality of library preparation and sequencing experiments.

**Major outputs when running this step for the testing example:**

* CPG\_1a.assembly.mt\_improved.fa # The improved genome assembly with better processing (re-assembling and circularization) of the mitochondrial genome.
* CPG\_1a.assembly.mt\_improved.chrMT.filter.pdf # The dotplot for the comparison between the improved mitochondrial genome assembly and the reference mitochondrial genome. You should see improved collinearity in this plot when compared with the similar plot generated in Step 15.

1. Generate the assembly modification list file for performing the final chromosome assignment by running the following commands:

$ cd ./../07.Supervised\_Final\_Assembly

$ bash LRSDAY.07.Supervised\_Final\_Assembly.1.sh

1. Edit the generated assembly modification list file (CPG\_1a.assembly.modification.list for the testing example) based on the genome-wide dotplot generated in Step 15 and the centromere profiles generated in Step 17. The modification list file consists of three comma-separated columns, which correspond to the original sequence name, sequence orientation, and new sequence name respectively. With this file, you can do three types of editing:
   * 1. If you need to change the current sequence order, you can move the corresponding rows upward or downward to reflect the correct order.
     2. If you need to invert the orientation of a given sequence, you can change its orientation from “+” to “-” in column 2.
     3. If you need to rename a given sequence, you can specify the new name in the third column.

For this testing example here, we need to move the row “chrIX,+,chrIX” downward to place it after the row “chrVIII,+,chrVIII”, so that chrIX will be placed after chrVIII in the final assembly. Also, we need to change the row “CPG\_1a.chrMT,+,CPG\_1a.chrMT” to “CPG\_1a.chrMT,+,chrMT” for renaming the assembled sequence corresponding to the mitochondrial genome. Finally, for all unassigned contigs (i.e., tig00000019\_pilon\_pilon\_pilon, tig00000021\_pilon\_pilon\_pilon, tig00000023\_pilon\_pilon\_pilon, tig00000024\_pilon\_pilon\_pilon for our testing example), since we have known that none of them represent major chromosomes by checking the dotplot of the scaffolded-assembly generated in Step 15, we can safely remove those lines completely.

**[Important Note]**

Although seemed trivial, it is important to replace all the “|” characters appearing in the sequence names of the final genome assembly with “\_” to prevent any error that might be related to this in your downstream analysis (not only from LRSDAY).

**Major outputs when running this step for the testing example:**

* CPG\_1a.assembly.modification.list # The assembly modification list file for manual editing to guide the final genome assembly.

1. Once all the modifications have been specified, run the following bash script to generate the final genome assembly as well as the associated genome-wide dotplot (Figure 9), assembly statistics (Table 1), and VCF files:

$ bash LRSDAY.07.Supervised\_Final\_Assembly.2.sh

Chart, scatter chart

Description automatically generated

***Figure 9. The dotplot comparison between the CPG\_1a final assembly and the S. cerevisiae reference genome.***

**Major outputs when running this step for the testing example:**

* CPG\_1a.assembly.final.fa # The final genome assembly generated by LRSDAY.
* CPG\_1a.assembly.final.filter.pdf # The genome-wide dotplot for the comparison between the final genome assembly and the reference genome.
* CPG\_1a.assembly.final.stats.txt # The summary table reporting basic assembly statistics of the final genome assembly. Users can compare this file with the similar file generated in Step 12 and Step 15. We also summarized such comparison in Table 1.
* CPG\_1a.assembly.final.filter.mummer2vcf.SNP.vcf # The VCF file showing SNP differences between the final genome assembly and the reference genome. This file can be used for assessing the final assembly quality.
* CPG\_1a.assembly.final.filter.mummer2vcf.INDEL.vcf # The VCF file showing INDEL differences between the final genome assembly and the reference genome. This file can be used for assessing the final assembly quality.

1. Re-run centromere annotation for the final genome assembly using the following commands:

$ cd ./../08.Centromere\_Annotation

$ bash LRSDAY.08.Centromere\_Annotation.sh

**Major outputs when running this step for the testing example:**

* CPG\_1a.nuclear\_genome.centromere.gff3 # The centromere annotation for the final genome assembly.

1. (Optional) Customize the configuration file for gene annotation. When running LRSDAY with your own data, edit the configuration file $LRSDAY\_HOME/misc/maker\_opts.customized.ctl if your sequenced organisms is neither *S. cerevisiae* nor *S. paradoxus* (See Appendix 2 for the details). If your sequenced organism is *S. cerevisiae* or *S. paradoxus*, no customization is needed unless you have native transcriptome or expressed sequence tag (EST) data for the strain that you sequenced. In this case, you can edit the line 16 of this file to provide the full path of the native transcriptome or EST assembly for your sequenced strain.
2. Annotate nuclear protein-coding genes and tRNAs for the final genome assembly, using the following commands. This step can be run with multiple threads.

$ cd ./../09.Nuclear\_Gene\_Annotation

$ bash LRSDAY.09.Nuclear\_Gene\_Annotation.sh

**Major outputs when running this step for the testing example:**

* CPG\_1a.nuclear\_genome.gff3 # The gene and tRNA annotation for the nuclear genome.
* CPG\_1a.nuclear\_genome.cds.fa # The CDSs of the annotated nuclear protein-coding genes.
* CPG\_1a.nuclear\_genome.trimmed\_cds.fa # The CDSs of the annotated nuclear protein-coding genes with the out-of-frame parts trimmed.
* CPG\_1a.nuclear\_genome.trimmed\_cds.log # The log file of the CDS trimming for the nuclear protein-coding gene annotation.
* CPG\_1a.nuclear\_genome.pep.fa # The translated protein sequences of the trimmed CDSs derived from the nuclear protein-coding gene annotation.
* CPG\_1a.nuclear\_genome.manual\_check.list # The list of suspicious nuclear gene annotations for manual curation.
* CPG\_1a.nuclear\_genome.PoFF.gff # The gene synteny information derived from CPG\_1a.nuclear\_genome.gff3, which will be used for Task 14 (Step 29).
* CPG\_1a.nuclear\_genome.PoFF.ffn # Same as CPG\_1a.nuclear\_genome.trimmed\_cds.fa but with simpler sequence IDs, which could be used for Task 14 (Step 34).
* CPG\_1a.nuclear\_genome.PoFF.ffa # Same as CPG\_1a.nuclear\_genome.pep.fa but with simpler sequence IDs, which will be used for Task 14 (Step 29).

1. Perform dedicated mitochondrial protein-coding and RNA annotation. If you are interested in studying mitochondrial genomes, we highly recommend running dedicated mitochondrial feature annotation with specialized software such as MFannot (<http://megasun.bch.umontreal.ca/cgi-bin/mfannot/mfannotInterface.pl>). Starting with v1.2.0, LRSDAY ships a local installation of MFannot that is ready to be used. So there is no need to run MFannot via its web portal anymore. Be sure to specify the correct genetic code table (e.g., “3” for annotating yeast mitochondrial genomes) for your analysis. Run MFannot locally by typing:

$ cd ./../10.Mitochondrial\_Gene\_Annotation

$ bash LRSDAY.10.Mitochondrial\_Gene\_Annotation.sh

**Major outputs when running this step for the testing example:**

* CPG\_1a.mitochondrial\_genome.gff3 # The gene and tRNA annotation for the mitochondrial genome.
* CPG\_1a. mitochondrial\_genome.cds.fa # The CDSs of the annotated mitochondrial protein-coding genes.
* CPG\_1a. mitochondrial\_genome.trimmed\_cds.fa # The CDSs of the annotated mitochondrial protein-coding genes with the out-of-frame parts trimmed.
* CPG\_1a.mitochondrial\_genome.trimmed\_cds.log # The log file of the CDS trimming for the mitochondrial protein-coding gene annotation.
* CPG\_1a.mitochondrial\_genome.pep.fa # The translated protein sequences of the trimmed CDSs derived from the mitochondrial protein-coding gene annotation.
* CPG\_1a.mitochondrial\_genome.manual\_check.list # The list of suspicious mitochondrial gene annotations for manual curation.
* CPG\_1a.mitochondrial\_genome.PoFF.gff # The gene synteny information derived from CPG\_1a.mitochondrial\_genome.gff3, which will be used for Task 14 (Step 29).
* CPG\_1a.mitochondrial\_genome.PoFF.ffn # Same as CPG\_1a. mitochondrial\_genome.trimmed\_cds.fa but with simpler sequence IDs, which could be used for Task 14 (Step 34).
* CPG\_1a.mitochondrial\_genome.PoFF.ffa # Same as CPG\_1a. mitochondrial\_genome.pep.fa but with simpler sequence IDs, which will be used for Task 14 (Step 29).

1. Annotate transposable elements (TEs) for the final genome assembly. This step can be run with multiple threads using the following commands:

$ cd ./../11.TE\_Annotation

$ bash LRSDAY.11.TE\_Annotation.sh

**Major outputs when running this step for the testing example:**

* CPG\_1a.nuclear\_genome.TE.gff3 # The TE annotation for the nuclear genome.

1. Annotate yeast telomere-associated core X elements for the final genome assembly, using the following commands:

$ cd ./../12.Core\_X\_Element\_Annotation

$ bash LRSDAY.12.Core\_X\_Element\_Annotation.sh

**Major outputs when running this step for the testing example:**

* CPG\_1a.nuclear\_genome.X\_element.gff3 # The core X-element annotation for the nuclear genome.

1. Annotate yeast telomere-associated Y’ elements for the final genome assembly by using the following commands:

$ cd ./../13.Y\_Prime\_Element\_Annotation

$ bash LRSDAY.13.Y\_Prime\_Element\_Annotation.sh

**Major outputs when running this step for the testing example:**

* CPG\_1a.nuclear\_genome.Y\_prime\_element.gff3 # The Y’-element annotation for the nuclear genome.

1. Perform orthology identification for protein coding genes by using the following commands:

$ cd ./../14.Gene\_Orthology\_Identification

$ bash LRSDAY14.Gene\_Orthology\_Identification.sh

In this step, a gene orthology relationship list is created between the annotated proteome and the SGD *S. cerevisiae* reference proteome based on both sequence similarity and synteny conservation. Based on this list, we further attach SGD systematic names to our gene annotation as shown in the “Name=” field of the generated GFF3 file (CPG\_1a.nuclear\_genome.SGD\_orthology\_mapped.gff3 and CPG\_1a.mitochondrial\_genome.SGD\_orthology\_mapped.gff3 for the testing example). For a given annotated gene, when more than one orthologous gene can be found in the SGD reference proteome, we will label all of its co-orthologs in the “Name=” filed with “/” between the alternative SGD systematic names (e.g., “YAR071W/YHR215W”), whereas when no orthologous gene can be found, we will label its gene name as “Name=NA”. This step can be run with multiple threads.

**Major outputs when running this step for the testing example:**

* CPG\_1a.nuclear\_genome.SGD\_orthology\_mapped.gff3 # The nuclear gene annotation with SGD systematic name tags.
* CPG\_1a.mitochondrial\_genome.SGD\_orthology\_mapped.gff3 # The mitocondrial gene annotation with SGD systematic name tags.

1. Integrate the annotation of different genomic features into a unified GFF3 file by using the following commands:

$ cd ./../15.Annotation\_Integration

$ bash LRSDAY.15.Annotation\_Integration.sh

**Major outputs when running this step for the testing example:**

* CPG\_1a.nuclear\_genome.tidy.fa # The final nuclear genome assembly.
* CPG\_1a.nuclear\_genome.tidy.gff3 # The final integrated annotation for the nuclear genome.
* CPG\_1a.nuclear\_genome.tidy.cds.fa # The CDSs of the final nuclear protein-coding gene annotation without out-of-frame trimming.
* CPG\_1a.nuclear\_genome.tidy.trimmed\_cds.fa # The CDSs of the final nuclear protein-coding gene annotation with the out-of-frame parts trimmed.
* CPG\_1a.nuclear\_genome.tidy.pep.fa # The translated protein sequences of the trimmed CDSs derived from the final nuclear protein-coding gene annotation.
* CPG\_1a.mitochondrial\_genome.tidy.fa # The final mitochondrial genome assembly.
* CPG\_1a.mitochondrial\_genome.tidy.gff3 # The final integrated annotation for the mitochondrial genome.
* CPG\_1a.mitochondrial\_genome.tidy.cds.fa # The CDSs of the final mitochondrial protein-coding gene annotation without out-of-frame trimming.
* CPG\_1a.mitochondrial\_genome.tidy.trimmed\_cds.fa # The CDSs of the final mitochondrial protein-coding gene annotation with the out-of-frame parts trimmed.
* CPG\_1a.mitochondrial\_genome.tidy.pep.fa # The translated protein sequences of the trimmed CDSs derived from the final mitochondrial protein-coding gene annotation.

# REFERENCES

1. Goffeau, A. *et al.* Life with 6000 Genes. *Science* **274**, 546–567 (1996).

2. Gordon, D. *et al.* Long-read sequence assembly of the gorilla genome. *Science (New York, N.Y.)* **352**, aae0344 (2016).

3. VanBuren, R. *et al.* Single-molecule sequencing of the desiccation-tolerant grass Oropetium thomaeum. *Nature* **527**, 508–11 (2015).

4. Bickhart, D. M. *et al.* Single-molecule sequencing and chromatin conformation capture enable de novo reference assembly of the domestic goat genome. *Nature Genetics* **49**, 643–650 (2017).

5. Badouin, H. *et al.* The sunflower genome provides insights into oil metabolism, flowering and Asterid evolution. *Nature* **546**, 148–152 (2017).

6. Jain, M. *et al.* Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature Biotechnology* **36**, 338–345 (2018).

7. Yue, J.-X. *et al.* Contrasting evolutionary genome dynamics between domesticated and wild yeasts. *Nature Genetics* **49**, 913–924 (2017).

8. Goodwin, S. *et al.* Oxford Nanopore sequencing, hybrid error correction, and de novo assembly of a eukaryotic genome. *Genome Research* **25**, 1750–1756 (2015).

9. McIlwain, S. J. *et al.* Genome sequence and analysis of a stress-tolerant, wild-derived strain of Saccharomyces cerevisiae used in biofuels research. *G3 (Bethesda, Md.)* **6**, 1757–66 (2016).

10. Istace, B. *et al.* de novo assembly and population genomic survey of natural yeast isolates with the Oxford Nanopore MinION sequencer. *GigaScience* **6**, 1–13 (2017).

11. Giordano, F. *et al.* De novo yeast genome assemblies from MinION, PacBio and MiSeq platforms. *Scientific reports* **7**, 3935 (2017).

12. Bergman, C. M. Horizontal transfer and proliferation of Tsu4 in Saccharomyces paradoxus. *Mobile DNA* **9**, 18 (2018).

13. Koren, S. *et al.* Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nature Biotechnology* **30**, 693–700 (2012).

14. Chin, C.-S. *et al.* Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data. *Nature methods* **10**, 563–9 (2013).

15. Berlin, K. *et al.* Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nature biotechnology* **33**, 623–630 (2015).

16. Chin, C.-S. *et al.* Phased diploid genome assembly with single-molecule real-time sequencing. *Nature Methods* **13**, 1050–1054 (2016).

17. Li, H. Minimap and miniasm: Fast mapping and de novo assembly for noisy long sequences. *Bioinformatics* **32**, 2103–2110 (2016).

18. Koren, S. *et al.* Canu: Scalable and accurate long-read assembly via adaptive κ-mer weighting and repeat separation. *Genome Research* **27**, 722–736 (2017).

19. Salamov, A. A. & Solovyev, V. V. Ab initio gene finding in Drosophila genomic DNA. *Genome Research* **10**, 516–522 (2000).

20. Stanke, M., Steinkamp, R., Waack, S. & Morgenstern, B. AUGUSTUS: A web server for gene finding in eukaryotes. *Nucleic Acids Research* **32**, W309–W312 (2004).

21. Besemer, J. & Borodovsky, M. GeneMark: Web software for gene finding in prokaryotes, eukaryotes and viruses. *Nucleic Acids Research* **33**, (2005).

22. Proux-Wéra, E., Armisén, D., Byrne, K. P. & Wolfe, K. H. A pipeline for automated annotation of yeast genome sequences by a conserved-synteny approach. *BMC Bioinformatics* **13**, 237 (2012).

23. Louis, E. J. & Haber, J. E. The structure and evolution of subtelomeric Y’ repeats in Saccharomyces cerevisiae. *Genetics* **131**, 559–574 (1992).

24. Strope, P. K. *et al.* The 100-genomes strains, an *S. cerevisiae* resource that illuminates its natural phenotypic and genotypic variation and emergence as an opportunistic pathogen. *Genome Research* **25**, 762–774 (2015).

25. Almeida, P. *et al.* A population genomics insight into the Mediterranean origins of wine yeast domestication. *Molecular ecology* **24**, 5412–5427 (2015).

26. Gallone, B. *et al.* Domestication and divergence of Saccharomyces cerevisiae beer yeasts. *Cell* **166**, 1397-1410.e16 (2016).

27. Peter, J. *et al.* Genome evolution across 1,011 Saccharomyces cerevisiae isolates. *Nature* **556**, 339–344 (2018).

28. Bergström, A. *et al.* A high-definition view of functional genetic variation from natural yeast genomes. *Molecular Biology and Evolution* **31**, 872–888 (2014).

29. Drillon, G., Carbone, A. & Fischer, G. SynChro: A fast and easy tool to reconstruct and visualize synteny blocks along eukaryotic chromosomes. *PLoS ONE* **9**, (2014).

30. Wang, Y. *et al.* MCScanX: A toolkit for detection and evolutionary analysis of gene synteny and collinearity. *Nucleic Acids Research* **40**, (2012).

31. O’Donnell, S. *et al.* 142 telomere-to-telomere assemblies reveal the genome structural landscape in Saccharomyces cerevisiae. 2022.10.04.510633 Preprint at https://doi.org/10.1101/2022.10.04.510633 (2022).

32. Smit, A., Hubley, R. & Green, P. RepeatMasker Open-4.0. 2013-2015 . *http://www.repeatmasker.org* Preprint at (2013).

33. Alonge, M. *et al.* Automated assembly scaffolding using RagTag elevates a new tomato system for high-throughput genome editing. *Genome Biology* **23**, 258 (2022).

34. Kolmogorov, M., Raney, B., Paten, B. & Pham, S. Ragout - A reference-assisted assembly tool for bacterial genomes. *Bioinformatics* **30**, (2014).

35. Pereira, V. Automated paleontology of repetitive DNA with REANNOTATE. *BMC genomics* **9**, 614 (2008).

36. Lowe, T. M. & Eddy, S. R. A computational screen for methylation guide snoRNAs in yeast. *Science (New York, N.Y.)* **283**, 1168–71 (1999).

37. Holt, C. & Yandell, M. MAKER2: an annotation pipeline and genome-database management tool for second-generation genome projects. *BMC bioinformatics* **12**, 491 (2011).

# APPENDIX

### *Appendix 1: Pre-shipped supporting data for LRSDAY*

We ship the following supporting datasets for the automatic execution of LRSDAY. Unless labeled otherwise, these datasets were either described or generated in our previous study7.

1. ATP6.cds.fa # The coding sequence (CDS) of the *S. cerevisiae* S288C *ATP6* gene.
2. fuzzy\_defragmentation.txt # the fuzzy defragmentation file for REannotate35.
3. Proteome\_DB\_for\_annotation.CDhit\_I95.fa # our curated proteome dataset for *S. cerevisiae* and other closely related yeast *sensu stricto* species.
4. query.Y\_prime\_element.long.fa # The representative *S. cerevisiae* Y’ element sequence.
5. S288C.ASM205763v1.fa.gz # the *S. cerevisiae* S288C genome assembly.
6. S288C.ASM205763v1.noncore\_masked.fa.gz # the *S. cerevisiae* S288C genome assembly with subtelomeres and chromosome-ends hard-masked.
7. S288C.centromere.fa # the centromere sequence of *S. cerevisiae* S288C.
8. S288C.gene.hmm # the hidden Markov model (HMM) for *de novo* gene annotation based on *S. cerevisiae* S288C.
9. S288C.X\_element.hmm # the hidden Markov model (HMM) for the core X element annotation based on *S. cerevisiae* S288C.
10. Sc-meth.sites # the *S. cerevisiae* methylation sites (shipped with snoScan36).
11. Sc-rRNA.fa # the *S. cerevisiae* rRNA sequences (shipped with snoScan36).
12. SGDref.PoFF.faa # the proteinortho proteome file generated for the SGD reference genome.
13. SGDref.PoFF.ffn # the proteinortho CDS file generated for the SGD reference genome.
14. SGDref.PoFF.gff # the proteinortho gene order gff file generated for the SGD reference genome.
15. te\_proteins.fasta # protein sequences for genes encoded within TEs (shipped with MAKER37).
16. TY2\_specific\_region.fa # the sequence of a Ty2 specific regions for differentiating Ty1 and Ty2.
17. TY\_lib.v20221221.tidy.fa # a custom RepeatMasker library for Ty annotation in *S. cerevisiae* and *S. paradoxus*.
18. TY\_lib.v20221221.LTRonly.tidy.fa # representative Ty LTR sequences of *S. cerevisiae* and *S. paradoxus*.

### *Appendix 2: Tips for adapting LRSDAY to other eukaryotic organisms*

The backbone modules of LRSDAY can be easily adapted for other eukaryotic organisms. Here are some tips with regard to this:

1. For Task 01 (long-read-based genome assembly; Step 12), be sure to adjust the genome size parameter in line 13 of the bash script LRSDAY.01.Long-read-based\_Genome\_Assembly.sh based on the estimated genome size of the organism that you sequenced.
2. For Task 04 (reference-guided assembly scaffolding; Step 15), be sure to modify the bash script LRSDAY.04.Reference-guided\_Assembly\_Scaffolding.sh to provide the reference genome file of your sequenced organisms to guide the scaffolding and chromosome assignment. It is very likely that the chromosomal cores and subtelomeres of your reference genome have not been clearly defined. In such case, you can provide the same reference genome file for both the “ref\_genome\_raw=” and “ref\_genome\_noncore\_masked=” parameters. The “chrMT\_tag=” and “gap\_size=” parameters should also be adjusted for your own project.
3. For Task 06 (mitochondrial genome assembly improvement; Step 18), be sure to modify the “gene\_start=”, “ref\_genome\_raw=”, and “chrMT\_tag=” parameters in the bash script LRSDAY.06.Mitochondrial\_Genome\_Assembly\_Improvement.sh based on your own project.
4. While many of the genomic feature annotation tasks are *S. cerevisiae* and *S. paradoxus* specific, Task 09 (nuclear gene annotation; Step 24) can be adapted for any eukaryotic organism. In general, you will need to edit the lines 16, 22, 34, 36, 44, 45, and 68-71 in the $LRSDAY\_HOME/misc/maker\_opts.customized.ctl file to feed organism-specific parameters into MAKER. Also, please refer to MAKER’s own Wiki page (<http://weatherby.genetics.utah.edu/MAKER/wiki/index.php/Main_Page>) and protocols (<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4286374/>) for technical details and advanced usage. Similarly, you can learn more about EVM from its website (<http://evidencemodeler.github.io/>).
5. For Task 10 (mitochondrial gene annotation; Step 26), be sure to specify the correct genetic code table for the mitochondrial genome of the species that you are studying. See this link (<https://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi>) for the details. The default table used by LRSDAY is table 3 for yeast mitochondria.
6. While Task 11 (TE annotation; Step 28) is heavily tuned for yeasts, the same tools that we used here (RepeatMasker32 and REannotate35) can be used for any eukaryotic organism. We recommend reading their respective manuals (distributed with the corresponding software) for adapting these tools in your own study.
7. For Task 14 (gene orthology identification; Step 34), you need to edit the “ref\_PoFF\_faa=” and “ref\_PoFF\_gff=” parameters based on the reference gene annotation that you used. Please check ProteinOrtho’s manual (<https://www.bioinf.uni-leipzig.de/Software/proteinortho/manual.html>) for more details on required file formats. The pre-shipped Perl script prepare\_PoFFfaa\_simple.pl and prepare\_PoFFgff\_simple.pl in the $LRSDAY\_HOME/scripts directory should help for this. You can run these two scripts as follows:

$ source ./../../env.sh

$ perl $LRSDAY\_HOME/scripts/prepare\_PoFFffn\_simple.pl -i prefix.pep.fa -o prefix.PoFF.faa

$ perl $LRSDAY\_HOME/scripts/prepare\_PoFFgff\_simple.pl -i prefix.raw.gff -o prefix.PoFF.gff