LRSDAY

User Manual

LRSDAY: Long-read Sequencing Data Analysis for Yeasts

Release v1.6.0

(2019-10-03)

Author Contact:

Jia-Xing Yue (岳家兴)

Email: yuejiaxing@gmail.com

GitHub: yjx1217

Twitter: iAmphioxus

Website: http://www.iamphioxus.org

Table of Contents

INTRODUCTION 1

Background 1

Overview of the LRSDAY workflow 2

Comparison with other methods 4

Experimental Design 4

Limitations and potential adaptation 9

Expected improvements 9

CITATIONS 10

MATERIALS 10

Hardware, operating system and network 10

Software or library requirements 10

Input data 11

Example data 12

PROCEDURE 13

Download, install and configure LRSDAY ● ***TIMING*** ***<1.5 h*** 13

Run LRSDAY with the testing example ● ***TIMING*** ***< 58 h*** 14

? TROUBLESHOOTING 28

● **TIMING** 31

ANTICIPATED RESULTS 32

REFERENCES 38

APPENDIX 41

Appendix 1: Pre-shipped supporting data for LRSDAY 41

Appendix 2: Tips for adapting LRSDAY to other eukaryotic organisms 42

INTRODUCTION

Background

Twenty years ago, the genome sequence of the budding yeast *Saccharomyces cerevisiae* was published[1]. 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 results[2–6]. 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* strains[8–11]. 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 study[7]. 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*), 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 (Fig. 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.

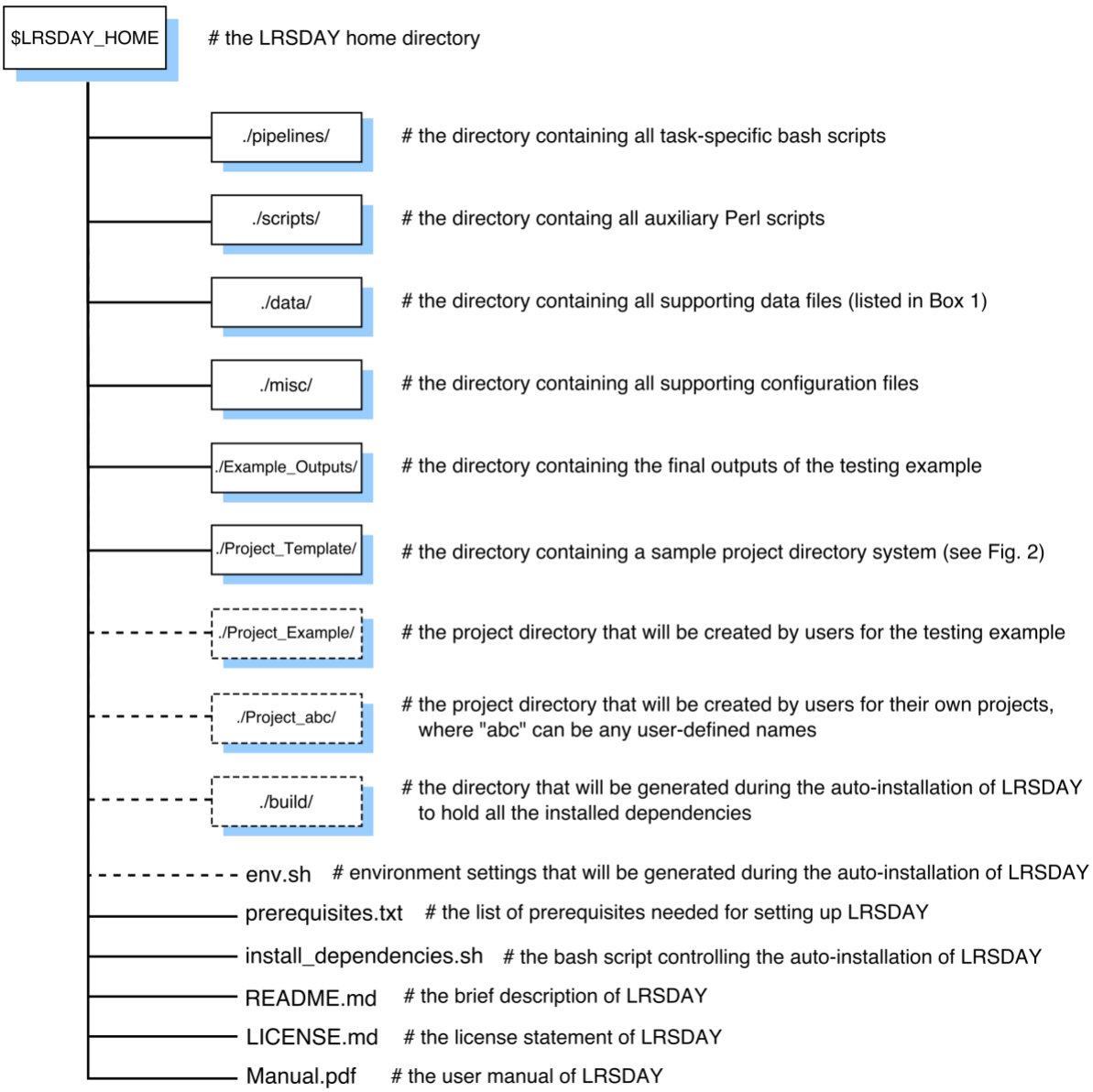

Figure 1. Overview of the LRSDAY directory system*. All the top-level directories (boxes, solid lines) and individual files of LRSDAY are listed and briefly described. Additional directories and files will be generated during the installation or execution of LRSDAY (boxes, dashed lines).*

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 organisms[8,12–17]. 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)[18–20]. Specifically for yeasts, a web-based gene annotation tool has been developed that combines both approaches[21]. 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 genomes[7] 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 (Fig. 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.

00. **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.
01. **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.
02. **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.
03. **Short-read-based_Assembly_Polishing** (Step 14; optional): When available, LRSDAY can make another round of 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.

04. **Reference-guided_Assembly_Scaffolding** (Step 15): 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.
05. **Centromere_Identity_Profiling** (Step 17): The pre-shipped *S. cerevisiae* centromere sequences are searched against the scaffolded assembly for chromosome-specific centromere identity profiling.
06. **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.
07. **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.
08. **Centromere_Annotation** (Step 22): The pre-shipped *S. cerevisiae* centromere sequences are searched against the final genome assembly for centromere annotation.
09. **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.
10. **Mitochondrial_Gene_Annotation** (Step 26): 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.
11. **TE_Annotation** (Step 28): The pre-shipped curated TE library (containing the long terminal repeats (LTRs) and internal sequences of the *S. cerevisiae* and *S.*

paradoxus Ty1-Ty5 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.

12. **Core_X_Element_Annotation** (Step 30): 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.
13. **Y_Prime_Element_Annotation** (Step 32): 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 forms[22], and we used a representative long-form Y' element as the query to maximize detection power.
14. **Gene_Orthology_Identification** (Step 34): 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.
15. **Annotation_Integration** (Step 35): 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.

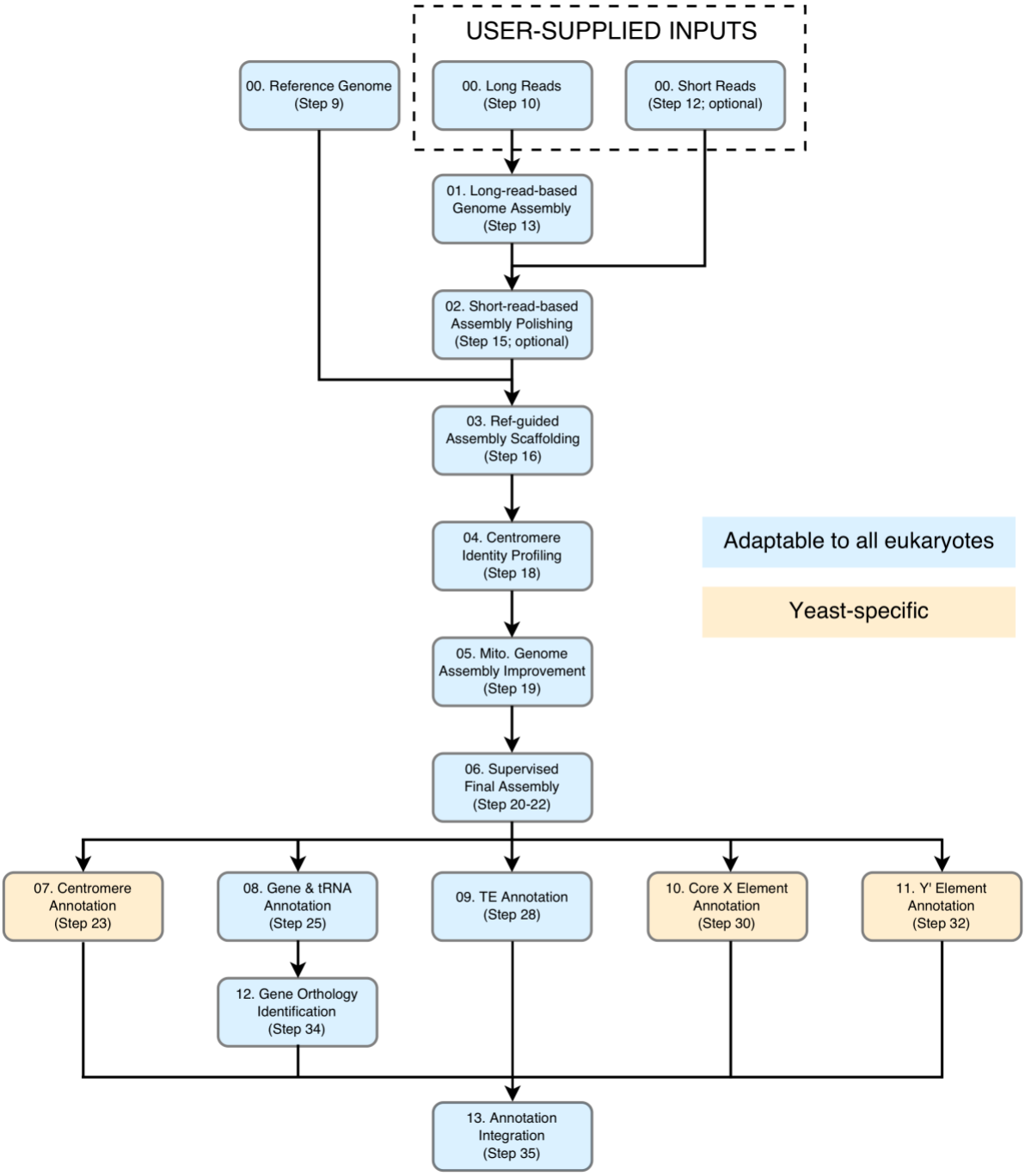

Figure 2. The workflow of LRSDAY. *Each box represents an individual module. These modules are numbered according to the tasks described in Experimental Design, with the corresponding protocol step numbers also indicated. Modules that can be adapted for other eukaryotes are colored in light blue while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

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* with a 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 sequencing[23–26], our knowledge of the overall genome content diversity[27] 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 Project[26]; 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. CHROnicle[28], MCScanX[29], 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. 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 Canu[17] 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)
- 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 v3.4 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 for PacBio Sequel 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 study[7]). 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 study[7] 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 the same study, which consists of both PacBio and Illumina reads produced from the *S. cerevisiae* strain SK1. The PacBio reads can be retrieved with the ENA analysis accession number ERZ448251 (https://www.ebi.ac.uk/ena/data/view/ERZ448251). The Illumina reads can be retrieved with the SRA sequencing run accession number SRR4074258 (https://www.ncbi.nlm.nih.gov/sra/?term=SRR4074258). In LRSDAY, we have provided bash scripts to automatically download and setup these data for the testing example.

PROCEDURE

Download, install and configure LRSDAY ● **TIMING <1 h**

1) Download the latest LRSDAY release (current version: v1.6.0) by entering the following commands in a terminal window:

   $ wget https://github.com/yjx1217/LRSDAY/releases/download/v1.6.0/LRSDAY-v1.6.0.tar.gz

   $ tar xvzf LRSDAY-v1.6.0.tar.gz

   $ cd LRSDAY-v1.6.0

   $ bash install_dependencies.sh

   ▲**CRITICAL STEP** Make sure your Internet connection is fast and stable when running this step since many tools will be downloaded here. Check that all the prerequisites (see *Software or library requirements* in EQUIPMENT or the prerequisite.txt file in the downloaded LRSDAY directory) have been installed on your system.

   ▲**CRITICAL STEP** Upon the successful completion of executing the bash script, you should see a confirmation message prompted out: “LRSDAY message: This bash script has been successfully processed! :)”. Otherwise, it means an error has occurred during the execution of the bash script, which interrupted the automatic installation process. This also applies to all the bash scripts used in Steps 8-35. Whenever an error is triggered, please check the error message and refer to the troubleshooting section when available. After fixing the cause of the error, re-run this step to initiate a new installation. The installer will prompt for the confirmation of deleting the build directory generated by the old run, always answer “yes” to authorize such action so that the new installation can start.

   ▲**CRITICAL STEP** Pay attention to the final message prompted by the installation script for the notes regarding to the additional manual setup step as well as the license restriction of some dependent tools for commercial users.

   ? TROUBLESHOOTING

2) 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

▲**CRITICAL STEP** While almost all required tools have been automatically installed and configured, manual configuration is needed for RepeatMasker[30]. 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.

? TROUBLESHOOTING

3) 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.

? TROUBLESHOOTING

4) 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 ● **TIMING < 46 h**

5) 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
```

▲**CRITICAL STEP** 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.

6) 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
```

7) Prepare the long 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, preferably also with a sequencing_summary.txt file generated during nanopore reads basecalling). Starting with v1.5.0, LRSDAY provides a dedicated bash script (LRSDAY.00.Nanopore_Reads_Basecalling_and_Demultiplexing.sh) for performing basecalling, demultiplexing, and read profile plotting of raw 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 download the PacBio RSII long reads (in both FASTQ and H5 formats) by entering:

```
$ cd ./../00.Long_Reads
$ bash LRSDAY.00.Retrieve_Sample_PacBio_Reads.sh
```

8) (Optional) Setup the long reads in their native formats. While FASTQ reads is self-sufficient for *de novo* assembly, long reads in their native formats (*.bax.h5 with the associated *.metadata.xml for PacBio RSII reads, *.bam for PacBio Sequel reads and *.fast5 for Oxford Nanopore reads) could be needed if you want to perform long-read-based assembly polishing (Step 13; when polisher="quiver", or "arrow" or "nanopolish").

   i) For PacBio RSII reads (as in our testing example):

The bax2bam file format conversion is needed (see https://github.com/PacificBiosciences/blasr/wiki/bax2bam-wiki:-installation,-basic-usage-and-FAQ for more details). To do this, first place the *.metadata.xml file(s) under the 00.Long_Reads/pacbio_fofn_files directory and place the *.bax.h5 files under the 00.Long_Reads/pacbio_fofn_files/Analysis_Results directory. Then setup the bax fofn file of those *.bax.h5 file(s) for each RSII SMRTcell (one RSII SMRTcell per fofn file) under the 00.Long_Reads/pacbio_fofn_files directory. For our testing example, these files should have already been set up automatically when running the LRSDAY.00.Retrieve_Sample_PacBio_Reads.sh script in Step 7. You can take it as an example to set up your own files. Once this is done, edit and run the provided script LRSDAY.00.PacBio.RSII_bax2bam.sh under the 00.Long_Reads directory to perform bax2bam conversion for each individual RSII SMRTcell. Do not mix bax files from different SMRTcells in your input bax fofn file when running this step. If you have run multiple SMRTcells for the same sample, do the bax2bam file conversion for each of them separately and then concatenate the resulting bam fofn files together to form a single merged bam fofn file that will be used for Step 13. For our testing example, under the 00.Long_Reads directory, edit the script LRSDAY.00.PacBio.RSII_bax2bam.sh for SK1.SMRTCell.1.RSII_bax.fofn, SK1.SMRTCell.2.RSII_bax.fofn, SK1.SMRTCell.3.RSII_bax.fofn, and SK1.SMRTCell.4.RSII_bax.fofn and run this script like below each time:

```
$ bash LRSDAY.00. PacBio.RSII_bax2bam.sh
```

This will result in four bam.fofn files (i.e. SK1.SMRTCell.1.bam.fofn, SK1.SMRTCell.2.bam.fofn, SK1.SMRTCell.3.bam.fofn, and SK1.SMRTCell.4.bam.fofn) written into the 00.Long_Reads/pacbio_fofn_files directory. Do the following to concatenate them into a single merged bam.fofn file:

```
$ cd pacbio_fofn_files
$ cat SK1.SMRTCell.*.bam.fofn > SK1.merged.bam.fofn
```

ii) For PacBio Sequel reads:

Place the bam files inside the 00.Long_Reads/pacbio_fofn_files directory first and prepare a single bam fofn file in the same directory.

iii) For Oxford Nanopore reads:

Just place the basecalled *.fast5 reads in the 00.Long_Reads/nanopore_basecalled_fast5_files directory. If you also have the sequencing_summary.txt file generated by the Oxford Nanopore Albacore or Guppy

basecaller during basecalling, also place this file in the same directory. If your Nanopore reads have not been basecalled, you can use our dedicated bash script LRSDAY.00.Nanopore_Reads_Basecalling_and_Demultiplexing.sh (shipped from v1.5.0) to perform basecalling, demultiplexing, and read profile plotting, in which case please put your raw *.fast5 reads in the Nanopore_raw_fast5_files directory.

9) (Optional) For your own data, you can 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 60X to reduce the computational consumption without much compromise in assembly quality:

```
$ bash LRSDAY.00.Long_Reads_Preprocessing.sh
```

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 subsampling_sequences.pl -i input.fq(.gz) -f fastq -s 0.1 -m random -p output
# randomly sampling the 10% sequences
```

or

```
perl subsampling_sequences.pl -i input.fq(.gz) -f fastq -s 0.1 -m longest -p output
# sampling the 10% longest sequences
```

▲CRITICAL STEP This step can be run with multiple threads to speed up the process. Depending on the CPU configuration of your Linux server/desktop, you can edit the “threads=” option in the bash script LRSDAY.00. Long_Reads_Preprocessing.sh to enable multi-threading. You can do the same for all the following tasks whenever the “threads=” option is provided in the corresponding task-specific bash script. Simple text editors such as emacs, vim, gedit or pico are recommended for such editing. Rich text editors might not work

▲CRITICAL STEP Note this step can take long to finish even with multi-threading, so we recommend running this step and all the other time-consuming steps (see the TIMING section) with “nohup” (https://en.wikipedia.org/wiki/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 &
```

10) (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
```

11) (Optional) Prepare short reads. 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
$ bash LRSDAY.00.Retrieve_Sample_Illumina_Reads.sh
```

? TROUBLESHOOTING

12) 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 (SK1.canu.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 (SK1.assembly.raw.filter.mummer2vcf.SNP.vcf and SK1.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.

▲CRITICAL STEP 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, Ra, 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-ra", 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, Ra, 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.

▲CRITICAL STEP 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", "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.

? TROUBLESHOOTING

13) (Optional) 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 the PacBio technology, this is done by using PacBio's own quiver/arrow pipeline[13]. If your long reads are generated by Oxford Nanopore sequencing, we recommend using racon-medaka (works better in our tests) or nanopolish or marginpolish. 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
```

? TROUBLESHOOTING

14) (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
```

TABLE 1 | Assembly statistics for the genome of *S. cerevisiae* strain SK1 assembled in the testing example.

Assembly statistics	**Raw assembly**	**Scaffolded assembly**	**Final assembly**
Total sequence count	39	38	34
Total sequence length (bp)	12524474	12530892	12473902
Min sequence length (bp)	12876	12836	12836
Max sequence length (bp)	1480272	1480337	1480337
Mean sequence length (bp)	321140.36	329760.32	366879.47
Median sequence length (bp)	26222.00	25986.50	57378.00
N50 (bp)	841779	923760	923760
L50	6	6	6
N90 (bp)	328097	328166	328166
L90	15	14	14
GC%	38.18	38.17	38.27
N%	0.00	0.04	0.04

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.

15) Perform chromosome-level scaffolding for the long-read-based assembly. Two scaffolders are supported: Ragout[31] and RaGOO[32]. For large genome (i.e. genome size > 100 Mb), please use RaGOO for scaffolding by setting scaffolder="ragoo". 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 (SK1.assembly.ref_based_scaffolded.stats.txt for this testing example) will be generated for the scaffolded assembly (Table 1).

    ▲CRITICAL STEP Please check the generated genome-wide dotplot (SK1.assembly.ref_based_scaffolded.filter.pdf for the testing example) (Fig. 3a) to verify the correctness of chromosomal identity assignment performed by Ragout or RaGOO 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. When doing scaffolding with ragout, you can check the generated AGP file (SK1.ragout.agp for the testing example) for the details of reference-based scaffolding.

    ▲CRITICAL STEP 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 (SK1.assembly.ref_based_scaffolded.chrMT.filter.pdf for the testing example) (Fig. 3a, inset). A list of such mitochondrial contigs will also be generated (SK1.assembly.ref_based_scaffolded.mt_contig.list for the testing example), which will be used in Step 18 for improving mitochondrial genome assembly.

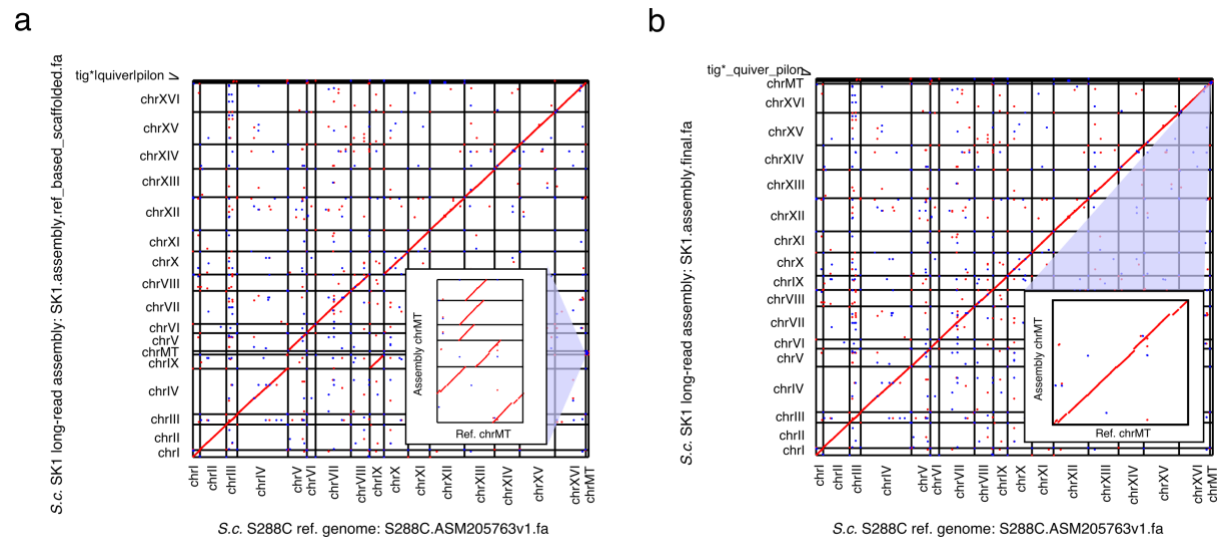

Figure 3. Genome-wide dotplots of the S. cerevisiae SK1 genome assembly generated in the LRSDAY testing example. *Both the raw scaffolded assembly (panel a; generated in Step 15) and the final assembly (panel b, generated in Step 21) are analyzed. The forward and reverse sequence matches are depicted in red and blue respectively, while the zoomed-in views of the mitochondrial genome (chrMT) comparison are shown in insets. In addition to the 16 nuclear chromosomes and the mitochondrial genome, the scaffolded and final assemblies also contain some unassigned short contigs (i.e. those whose names start with as "tig") that are derived from highly repetitive regions.*

16) (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. 100>

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.

17) 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

▲CRITICAL STEP 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.

18) Perform mitochondrial genome assembly improvement by running the following commands:

$ cd ./../06.Mitochondrial_Genome_Assembly_Improvement

$ bash LRSDAY.06.Mitochondrial_Genome_Assembly_Improvement.sh

▲CRITICAL STEP Check the generated final mitochondrial genome dotplot (SK1.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 (Fig. 3b, inset). 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.

? TROUBLESHOOTING

19) 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
```

20) Edit the generated assembly modification list file (SK1.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:

    i) If you need to change the current sequence order, you can move the corresponding rows upward or downward to reflect the correct order.

    ii) If you need to invert the orientation of a given sequence, you can change its orientation from "+" to "-" in column 2.

    iii) 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 "chrMT_Contig1,+,chrMT_Contig1" to "chrMT_Contig1,+,chrMT" for renaming the assembled sequence corresponding to the mitochondrial genome. Finally, for all unassigned contigs, updated their contig names by replacing "|" with "_". For example, the row ""tig00000065|quiver|pilon,+,tig00000065|quiver|pilon" will need to be changed to ""tig00000065|quiver|pilon,+,tig00000065_quiver_pilon". This particular change is to prevent potential problems when further processed with downstream tools that do not support "|" in sequence names (e.g. EVM that will be used in Step 24). For your own project, by referencing to the dotplots generated in Step 15 and 18, you can choose to remove those lines corresponding to unplaced contigs since they are very likely to be redundant.

    ▲CRITICAL STEP 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 for LRSDAY).

21) 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 (Fig. 3b), assembly statistics (Table 1), and VCF files:

```
$ bash LRSDAY.07.Supervised_Final_Assembly.2.sh
```

22) Re-run centromere annotation for the final genome assembly using the following commands:

    $ cd ./../08.Centromere_Annotation

    $ bash LRSDAY.08.Centromere_Annotation.sh

23) (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.

24) 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

25) (Optional) A manual checklist file (SK1.nuclear_genome.EVM.manual_check.list for the testing example) containing a list of genes with suspicious annotations will be generated in Step 24. As labeled in this file, these annotated gene models could be fragmented, frameshifted or containing internal stop codons. Potentially, these genes could be good candidates for pseudogenes. Manually inspect the annotated gene models of these genes by loading the annotation result (SK1.nuclear_genome.EVM.gff3 for the testing example) together with the protein/EST-alignment evidence files generated in Step 24 (SK1.nuclear_genome.protein_evidence.gff3 and SK1.nuclear_genome.est_evidence.gff3 for the testing example) into IGV[33] to check how well these suspicious gene models are supported by the corresponding protein/EST-alignment evidence and to tag or remove those truly problematic ones in your downstream analysis.

26) 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 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
```

27) (Optional) A manual checklist file (SK1.mitochondrial_genome.mfannot.manual_check.list for the testing example) containing a list of genes with suspicious annotations will be generated in Step 26. Manual gene boundary refinement based on CDS/protein sequence alignment is recommended for curating these potentially problematic gene models (e.g. cox2 for the testing example) based on the sequence alignment with reference mitochondrial genes.
28) 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
```

29) (Optional) TE activity can be highly dynamic in the genome with many complex cases such as fragmentation and nested insertion. In LRSDAY, we used REannotate[34] to automatically resolve these complex cases, which works well for most cases but it could occasionally misjoin two adjacent TEs when they are closely spaced. Further inspect and curate the LRSDAY TE annotation output (SK1.TE.gff3 for the testing example) by visualizing it in IGV[33] together with the raw REannotate annotation (SK1.REannotate.gff for the testing example). For each TE found in the LRSDAY TE annotation output, examine its corresponding LTR and internal region structure based on the raw REannotate annotation to check for misjoinings. If needed, you can manually edit the corresponding TE annotation output file to decouple the misjoinings.
30) 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
```

31) (Optional) Inspect the alignment file for annotated core X element. In LRSDAY, we label the identified core X elements to be “partial” if they are shorter than 300 bp. This should work for most cases but we recommend inspecting the generated alignment file (SK1.X_element.aln.fa for the testing example) for further curation. Upon the curation, manually adjust the “partial” labeling in the annotation file (SK1.X_element.gff3 for the testing example) when needed.

32) 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

33) (Optional) Inspect the alignment file for annotated Y' element. Like for the core X element annotation, we used a hard length cutoff (3500 bp) to label if the identified Y' elements are "partial". We recommend manually inspecting the generated alignment file (SK1.Y_prime_element.aln.fa for the testing example) to check if the "partial" labeling is needed and edit the annotation file (SK1.Y_prime_element.gff3 for the testing example) accordingly.

34) 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 (SK1.updated.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.

35) 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

? TROUBLESHOOTING

Troubleshooting advice can be found in Table 2.

TABLE 2 | Troubleshooting table.

Step	Problem	Possible reason	Solution
1	Downloading errors or unresponsive remote servers	Unstable internet connection or temporary problems of the remote servers where the tools for downloading are hosted.	Stabilizing the internet connection and rerun: bash install_dependencies.sh. You will be prompted for the question asking if want to delete the old “build” directory that created in your failed installation. Answer “yes”. Or you can delete the “build” directory manually first and then run: bash install_dependencies.sh
1	Compilation or installation error for specific tools.	Prerequisites not satisfied or corner cases due to your specific system settings.	If missing prerequisites are found (see *software and library requirements* in EQUIPMENT), install the missing prerequisites and give another try as described above. Otherwise, record the error message and email it to the developers of the problematic tools or the authors of this protocol for further problem diagnosis.
2	Cannot find the file “env.sh”.	Step 1 has failed.	Check the error message that you got when running Step 1 and refer to the troubleshooting for Step 1.
3	“echo $rmblast_dir” returns nothing.	Step 2 has failed or your terminal session got interrupted before this step.	Re-load environment settings in Step 2 and then re-try this step.
11	Downloading warnings/errors encountered.	Temporary SRA server problems.	Directly download the sample Illumina reads for the testing example by running the following commands:

			$ wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR407/008/SRR4074258/SRR4074258_1.fastq.gz
			$ ln -s SRR4074258_1.fastq.gz SRR4074258_pass_1.fastq.gz
			$ wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR407/008/SRR4074258/SRR4074258_2.fastq.gz
			$ ln -s SRR4074258_2.fastq.gz SRR4074258_pass_2.fastq.gz

12	The total assembly size is much smaller than the expected value or the assembly is too fragmented.	Insufficient sequencing depth of coverage.	Obtain more reads. We recommend a minimal sequencing depth of 20X.
12	Assembly runs too slow.	This likely will happen for Canu with high-depth Nanopore reads.	Three options are available: 1) Use the pre-shipped bash script LRSDAY.00.Long_Reads_Preprocessing.sh in the 00.Long_Reads subdirectory or the pre-shipped Perl script subsampling_seqeunces.pl in the $LRSDAY_HOME/scripts directory to downsample the reads to desired coverage of depth. 2) Use the customized parameters as suggested in the bash script LRSDAY.01.Long-read-based_Genome_Assembly.sh 3) Use alternative assemblers such as Flye or shasta for this step.
13	Error encountered for long-read-based polishing.	The natively formatted long-reads have not been correctly set up.	Check Step 8 carefully to make sure all instructions have been followed. Walking

			through the Step 8 for the testing example should help.
18	Only partial assembly is obtained for the mitochondrial genome.	High AT and repeat content of the mitochondrial genome is challenging for *de novo* genome assembly.	If the sequencing is done with PacBio, obtaining more reads should solve the problem. If sequencing is done with Oxford Nanopore data, currently there is no satisfying solution. Future improvement of the Oxford Nanopore sequencing technology will likely solve this problem given the rapid development of this technology.

● TIMING

The following timing information was measured on a Linux computing server with an Intel Xeon CPU E5-2630L v3 (1.80 GHz) using 4 threads. Enabling multithreading can substantially decrease the processing time. All the following steps were processed when measuring the computation time.

Steps 1-4, set up LRSDAY: < 1.5 h.

Step 5, prepare the project directory for the testing example: 5 s.

Step 6, prepare the reference genomes for the testing example: 1 s.

Step 7, prepare the long reads for the testing example: 8 h 20 min.

Step 8 (optional), set up the natively formatted long reads for Step 13: < 1 h.

Step 9 (optional), downsampling the long reads for Step 13: 10 min.

Step 11 (optional), prepare the Illumina reads for the testing example: 20 min.

Step 12, *de novo* assembly using long-reads: 6 h .

Step 13 (optional), assembly polishing using long reads: 7 h.

Step 14 (optional), assembly polishing using Illumina reads: 1 h 30 min.

Step 15, reference-guided scaffolding for the raw assembly: 12 min.

Step 17, centromere identity profiling for the scaffolded assembly: 1 s.

Step 18, mitochondrial genome assembly improvement: 4 min.

Step 19, generate the assembly modification list file: 1 s.

Step 20, manual editing the assembly modification list file: 30 s.

Step 21, finalize genome assembly based on the assembly modification list file: 20 s.

Step 22, centromere annotation for the final assembly: 1 s.

Step 24, protein-coding gene and tRNA annotation for the final assembly: 20 h.

Step 26, mitochondrial gene and RNA annotation for the final assembly: 3 min.

Step 28, TE annotation for the final assembly: 5 min.

Step 30, core X element annotation for the final assembly: 2 min.

Step 32, Y' element annotation for the final assembly: 2 min 30 s.

Step 34, gene orthology identification for the annotated protein-coding genes: 20 min.

Step 35, final annotation integration: 40 min.

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. As demonstrated in Fig. 3b, the final genome assembly is highly continuous, with each chromosome assembled in an end-to-end fashion. Both genome-wide dotplots (like those shown in Fig. 3) and summary statistics (like those listed in Table 1) 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 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. In Table 3, we further summarized the major outputs of the testing example. The final genome assembly and annotation outputs generated with this testing example are also provided in the directory Example_Outputs for users to make direct comparison with their own results.

TABLE 3 | Major LRSDAY outputs from the testing example.

Task	Step	Output file or directory	Explanation for the output
01	12	SK1.assembly.raw.fa	The long-read-based *de novo* genome assembly containing all the contigs assembled by Canu[17]. This file is soft-linked to the SK1.assembly.canu.fa file.
		SK1.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. Users can compare this file with the similar file generated in Step 15 and Step 21. We also summarized such comparison in Table 1.
		SK1.assembly.raw.filter.pdf	The genome-wide dotplot for the comparison between the raw genome assembly and the reference genome.
		SK1.assembly.raw.filter.mummer2vcf.SNP.vcf	The VCF file showing SNP differences between the raw genome assembly and the reference genome. This file can be

			used for assessing the raw assembly quality.
		SK1.assembly.raw.filter.mummer2vcf.INDEL.vcf	The VCF file showing INDEL differences between the raw genome assembly and the reference genome. This file can be used for assessing the raw assembly quality.
		SK1_canu_out	The directory containing all the output files of Canu.
02	13	SK1.assembly.long_read_polished.fa	The long-read-based polished assembly generated by the PacBio quiver/arrow pipeline in FASTA format. This file is soft-linked to the file SK1.assembly.consensus.fa.
		SK1.assembly.consensus.fq.gz	The long-read-based polished assembly generated by the PacBio quiver/arrow pipeline in compressed FASTQ format.
		SK1.assembly.consensus.vcf.gz	The compressed VCF file reporting all the base-level changes performed in long-read-based polishing.
		SK1.pbalign.bam	The BAM file of long reads (PacBio reads) mapping against the input genome assembly. When running with Nanopore reads, the alignment file will be named as $prefix.minimap2.bam where $prefix is the file name prefix that you defined.
03	14	SK1.assembly.illumina_read_polished.fa	The polished genome assembly generated by Pilon[35].
		SK1.assembly.illumina_read_polished.vcf.gz	The compressed VCF file reporting the variants identified by Pilon based on short reads mapping against the input genome assembly.
		SK1.assembly.illumina_read_polished.changes	The space-delimited record of all the changes that Pilon made during the assembly polishing. The four columns are: the original sequence coordinate, the new sequence coordinate after the correction, the original base, the new base after the correction.
		SK1.realn.bam	The BAM file of short reads mapping against the input genome assembly. You can load this file into IGV together with the input genome assembly to evaluate the performance of Illumina-based error correction.
04	15	SK1.assembly.ref_based_scaffolded.fa	The scaffolded genome assembly based on the reference genome. This file is soft-linked to the SK1.ragout.fa file.
		SK1.assembly.ref_based_scaffolded.stats.txt	The summary table reporting basic assembly statistics of the scaffolded genome assembly. Users can compare this file with the similar file generated in Step 12 and Step 21. We also summarized such comparison in Table 1.

		SK1.ragout.agp	the AGP file reporting the order and orientation of each input contig used during scaffolding.
		SK1_ragout_out	The directory containing all the output files of Ragout[31].
		SK1.ragout.filter.pdf	The genome-wide dotplot for the comparison between the scaffolded assembly and the reference genome.
		SK1.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.
		SK1.assembly.ref_based_scaffolded.mt_contig.fa	The assembled contig sequences corresponding to the mitochondrial genome.
		SK1.assembly.ref_based_scaffolded.chrMT.filter.pdf	The dotplot for the comparison between the scaffolded mitochondrial genome assembly and the reference mitochondrial genome.
05	17	SK1.centromere.gff3	The profiled centromere identities for the scaffolded genome assembly.
06	18	SK1.assembly.mt_improved.fa	The improved genome assembly with better processing (re-assembling and circularization) of the mitochondrial genome.
		SK1.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.
07	19-21	SK1.assembly.modification.list	The assembly modification list file for manual editing to guide the final genome assembly.
		SK1.assembly.final.fa	The final genome assembly generated by LRSDAY.
		SK1.assembly.final.filter.pdf	The genome-wide dotplot for the comparison between the final genome assembly and the reference genome.
		SK1.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.
		SK1.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.
		SK1.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.
08	22	SK1.centromere.gff3	The centromere annotation for the final genome assembly.

09	24	SK1.nuclear_genome.maker.raw.gff3	The raw MAKER[36] annotation for nuclear protein-coding genes and tRNA genes.
		SK1.nuclear_genome.EVM.gff3	The final EVM[37] annotation for nuclear protein-coding genes and tRNA genes with systematically assigned gene IDs.
		SK1.nuclear_genome.protein_evidence.gff3	The protein-to-genome alignment evidences generated by MAKER. This file can be used for manual curation of those suspicious annotations.
		SK1.nuclear_genome.est_evidence.gff3	The EST-to-genome alignment evidences generated by MAKER. This file can be used for manual curation of those suspicious annotations.
		SK1.nuclear_genome.EVM.cds.fa	The CDSs of the nuclear protein-coding genes annotated by EVM.
		SK1.nuclear_genome.EVM.trimmed_cds.fa	The CDSs of the nuclear protein-coding genes annotated by EVM with the out-of-frame parts trimmed.
		SK1.nuclear_genome.EVM.trimmed_cds.log	The log file of the CDS trimming for the final EVM nuclear protein-coding gene annotation.
		SK1.nuclear_genome.EVM.pep.fa	The translated protein sequences of the trimmed CDSs derived from the EVM nuclear protein-coding gene annotation.
		SK1.nuclear_genome.EVM.manual_check.list	The list of suspicious nuclear gene annotations for manual curation.
		SK1.nuclear_genome.EVM.PoFF.gff	The gene synteny information derived from SK1.nuclear_genome.EVM.gff3, which will be used for Task 14 (Step 34).
		SK1.nuclear_genome.EVM.PoFF.ffn	Same as SK1.nuclear_genome.EVM.trimmed_cds.fa but with simpler sequence IDs, which could be used for Task 14 (Step 34).
		SK1.nuclear_genome.EVM.PoFF.ffa	Same as SK1.EVM.pep.fa but with simpler sequence IDs, which will be used for Task 14 (Step 34).
10	26	SK1.assembly.mitochondrial_genome.fa	The mitochondrial genome sequence(s) in the final genome assembly.
		SK1.mitochondrial_genome.mfannot.out	The raw mitochondrial gene annotation output from mfannot.
		SK1.mitochondrial_genome.mfannot.gff3	The mitochondrial gene annotation in GFF3 format.
		SK1.mitochondrial_genome.mfannot.cds.fa	The CDSs of the mitochondrial protein-coding genes annotated by mfannot.
		SK1.mitochondrial_genome.mfannot.trimmed_cds.fa	The CDSs of the mitochondrial protein-coding genes annotated by mfannot with out-of-frame parts trimmed.
		SK1.mitochondrial_genome.mfannot.trimmed_cds.log	The log file of the CDS trimming for the final mfannot mitochondrial protein-coding gene annotation.
		SK1.mitochondrial_genome.mfannot.pep.fa	The translated protein sequences of the trimmed CDSs derived from the mfannot mitochondrial protein-coding genes annotation.
		SK1.mitochondrial_genome.mfannot.manual_check.list	The list of suspicious mitochondrial gene annotations for manual curation.

		SK1.mitochondrial_genome.mfannot.PoFF.gff	The gene synteny information derived from SK1.mitochondrial_genome.mfannot.gff3, which will be used for Task 14 (Step 34).
		SK1.mitochondrial_genome.mfannot.PoFF.ffn	Same as SK1.mitochondrial_genome.mfannot.trimmed_cds.fa but with simpler sequence IDs, which could be used for Task 14 (Step 34).
		SK1.mitochondrial_genome.mfannot.PoFF.ffa	Same as SK1.mitochondrial_genome.mfannot.pep.fa but with simpler sequence IDs, which will be used for Task 14 (Step 34).
11	28	SK1.REannotate.gff	The raw TE annotation from REannotate. This file can be used for further curating TE annotation.
		SK1.TE.gff3	The final TE annotation from LRSDAY.
12	30	SK1.X_element.gff3	The final core X element annotation from LRSDAY.
		SK1.X_element.fa	The sequences of all the annotated core X elements.
		SK1.X_element.aln.fa	The sequence alignment of all the annotated core X elements for further checking whether the annotated feature is complete or partial as well as whether this is consistent with the labeling in the annotation file SK1.X_element.gff3.
13	32	SK1.Y_prime_element.gff3	The final Y' element annotation from LRSDAY.
		SK1.Y_prime_element.fa	The sequences of all the annotated Y' elements.
		SK1.Y_prime_element.aln.fa	The sequence alignment of all the annotated Y' elements for further checking whether the annotated feature is complete or partial as well as whether this is consistent with the labeling in the annotation file SK1.Y_prime_element.gff3.
14	34	SK1.nuclear_gene.proteinortho	The nuclear gene orthology mapping between the annotated genes and the reference gene sets based only on sequence similarity.
		SK1.nuclear_gene.poff	The nuclear gene orthology mapping between the annotated genes and the reference gene sets based on both sequence similarity and synteny conservation.
		SK1.nuclear_gene.updated.gff3	The updated nuclear gene annotation with reference-based gene name labeling.
		SK1.mitochondrial_gene.proteinortho	The mitochondrial gene orthology mapping between the annotated genes and the reference gene sets based only on sequence similarity.
		SK1.mitochondrial_gene.poff	The mitochondrial gene orthology mapping between the annotated genes and the reference gene sets based on

			both sequence similarity and synteny conservation.
		SK1.mitochondrial_gene.updated.gff3	The updated mitochondrial gene annotation with reference-based gene name labeling.
15	35	SK1.assembly.final.fa	A copy of the final genome assembly generated in Task 07 (Step 21).
		SK1.final.gff3	The final integrated annotation from LRSDAY.
		SK1.final.cds.fa	The CDSs of the final protein-coding gene annotation without out-of-frame trimming.
		SK1.final.trimmed_cds.fa	The CDSs of the final protein-coding gene annotation with the out-of-frame parts trimmed.
		SK1.final.trimmed_cds.log	The log file of the CDS trimming for the final protein-coding gene annotation.
		SK1.final.pep.fa	The translated protein sequences of the trimmed CDSs derived from the final protein-coding gene annotation.
		SK1.final.manual_check.list	The list of suspicious gene annotations for manual curation.
		SK1.assembly.nuclear_genome.fa	The final nuclear genome assembly.
		SK1.nuclear_genome.trimmed_cds.fa	The CDSs of the final nuclear protein-coding gene annotation with the out-of-frame parts trimmed.
		SK1.nuclear_genome.trimmed_cds.log	The log file of the CDS trimming for the final nuclear protein-coding gene annotation.
		SK1.nuclear_genome.pep.fa	The translated protein sequences of the trimmed CDSs derived from the final nuclear protein-coding gene annotation.
		SK1.nuclear_genome.manual_check.list	The list of suspicious nuclear gene annotations for manual curation.
		SK1.assembly.mitochondrial_genome.fa	The final mitochondrial genome assembly.
		SK1.mitochodnrial_genome.trimmed_cds.fa	The CDSs of the final mitochondrial protein-coding gene annotation with the out-of-frame parts trimmed.
		SK1.mitochodnrial_genome.trimmed_cds.log	The log file of the CDS trimming for the final mitochondrial protein-coding gene annotation.
		SK1.mitochondrial_genome.pep.fa	The translated protein sequences of the trimmed CDSs derived from the final mitochondrial protein-coding gene annotation.
		SK1.mitochondrial_genome.manual_check.list	The list of suspicious mitochondrial gene annotations for manual curation.

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* **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. *Nat. Genet.* **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. *Nat. Biotechnol.* **36**, 338–345 (2018).
7. Yue, J.-X. *et al.* Contrasting evolutionary genome dynamics between domesticated and wild yeasts. *Nat. Genet.* **49**, 913–924 (2017).
8. Goodwin, S. *et al.* Oxford Nanopore sequencing, hybrid error correction, and de novo assembly of a eukaryotic genome. *Genome Res.* **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).* **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. *Sci. Rep.* **7**, 3935 (2017).
12. Koren, S. *et al.* Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nat. Biotechnol.* **30**, 693–700 (2012).
13. Chin, C.-S. *et al.* Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data. *Nat. Methods* **10**, 563–9 (2013).
14. Berlin, K. *et al.* Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nat. Biotechnol.* **33**, 623–630 (2015).
15. Chin, C.-S. *et al.* Phased diploid genome assembly with single-molecule real-time sequencing. *Nat. Methods* **13**, 1050–1054 (2016).
16. Li, H. Minimap and miniasm: Fast mapping and de novo assembly for noisy long

sequences. *Bioinformatics* **32**, 2103–2110 (2016).

17. Koren, S. *et al.* Canu: Scalable and accurate long-read assembly via adaptive κ-mer weighting and repeat separation. *Genome Res.* **27**, 722–736 (2017).
18. Salamov, A. A. & Solovyev, V. V. Ab initio gene finding in Drosophila genomic DNA. *Genome Res.* **10**, 516–522 (2000).
19. Stanke, M., Steinkamp, R., Waack, S. & Morgenstern, B. AUGUSTUS: A web server for gene finding in eukaryotes. *Nucleic Acids Res.* **32**, W309–W312 (2004).
20. Besemer, J. & Borodovsky, M. GeneMark: Web software for gene finding in prokaryotes, eukaryotes and viruses. *Nucleic Acids Res.* **33**, (2005).
21. 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).
22. Louis, E. J. & Haber, J. E. The structure and evolution of subtelomeric Y' repeats in Saccharomyces cerevisiae. *Genetics* **131**, 559–574 (1992).
23. 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 Res.* **25**, 762–774 (2015).
24. Almeida, P. *et al.* A population genomics insight into the Mediterranean origins of wine yeast domestication. *Mol. Ecol.* **24**, 5412–5427 (2015).
25. Gallone, B. *et al.* Domestication and divergence of Saccharomyces cerevisiae beer yeasts. *Cell* **166**, 1397-1410.e16 (2016).
26. Peter, J. *et al.* Genome evolution across 1,011 Saccharomyces cerevisiae isolates. *Nature* **556**, 339–344 (2018).
27. Bergström, A. *et al.* A high-definition view of functional genetic variation from natural yeast genomes. *Mol. Biol. Evol.* **31**, 872–888 (2014).
28. 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).
29. Wang, Y. *et al.* MCScanX: A toolkit for detection and evolutionary analysis of gene synteny and collinearity. *Nucleic Acids Res.* **40**, (2012).
30. Smit, A., Hubley, R. & Green, P. RepeatMasker Open-4.0. 2013-2015 . *http://www.repeatmasker.org* (2013).
31. Kolmogorov, M., Raney, B., Paten, B. & Pham, S. Ragout - A reference-assisted assembly tool for bacterial genomes. *Bioinformatics* **30**, (2014).
32. Alonge, M. *et al.* Fast and accurate reference-guided scaffolding of draft genomes.

bioRxiv (2019). doi:10.1101/519637

33. Robinson, J. T. *et al.* Integrative genomics viewer. *Nat. Biotechnol.* **29**, 24–26 (2011).
34. Pereira, V. Automated paleontology of repetitive DNA with REANNOTATE. *BMC Genomics* **9**, 614 (2008).
35. Walker, B. J. *et al.* Pilon: An integrated tool for comprehensive microbial variant detection and genome assembly improvement. *PLoS One* **9**, (2014).
36. Holt, C. & Yandell, M. MAKER2: an annotation pipeline and genome-database management tool for second-generation genome projects. *BMC Bioinformatics* **12**, 491 (2011).
37. Haas, B. J. *et al.* Automated eukaryotic gene structure annotation using EVidenceModeler and the program to assemble spliced alignments. *Genome Biol.* **9**, R7 (2008).
38. Lowe, T. M. & Eddy, S. R. A computational screen for methylation guide snoRNAs in yeast. *Science* **283**, 1168–71 (1999).
39. Minkin, I., Patel, A., Kolmogorov, M., Vyahhi, N. & Pham, S. Sibelia: A scalable and comprehensive synteny block generation tool for closely related microbial genomes. in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **8126 LNBI**, 215–229 (2013).

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 study[7].

1) ATP6.cds.fa # The coding sequence (CDS) of the *S. cerevisiae* S288C *ATP6* gene.
2) fuzzy_defragmentation.txt # the fuzzy defragmentation file for REannotate[34].
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 snoScan[38]).
11) Sc-rRNA.fa # the *S. cerevisiae* rRNA sequences (shipped with snoScan[38]).
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 MAKER[36]).
16) TY2_specific_region.fa # the sequence of a Ty2 specific regions for differentiating Ty1 and Ty2.
17) TY_lib.Yue_et_al_2017_NG.fa # a custom RepeatMasker library for Ty annotation in *S. cerevisiae* and *S. paradoxus*.
18) TY_lib.Yue_et_al_2017_NG.LTRonly.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) By default, Task 04 (reference-guided assembly scaffolding; Step 15) performs reference-guided scaffolding using the whole genome alignment constructed by Sibelia[39]. Sibelia is designed for processing small genomes (<100 Mb). For organisms with large genomes, you should install Progressive Cactus (https://github.com/glennhickey/progressiveCactus) and use it to build the whole genome alignment in HAL format and feed it directly to Ragout[31]. Please refer to Ragout's user manual (http://fenderglass.github.io/Ragout/usage.html) for this advanced usage. We have pre-shipped the HAL tools (https://github.com/ComparativeGenomicsToolkit/hal) to enable Ragout to process the HAL file generated by Progressive Cactus.
4) 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.
5) 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/).

6) 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.
7) While Task 11 (TE annotation; Step 28) is heavily tuned for yeasts, the same tools that we used here (RepeatMasker[30] and REannotate[34]) 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.
8) 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
```