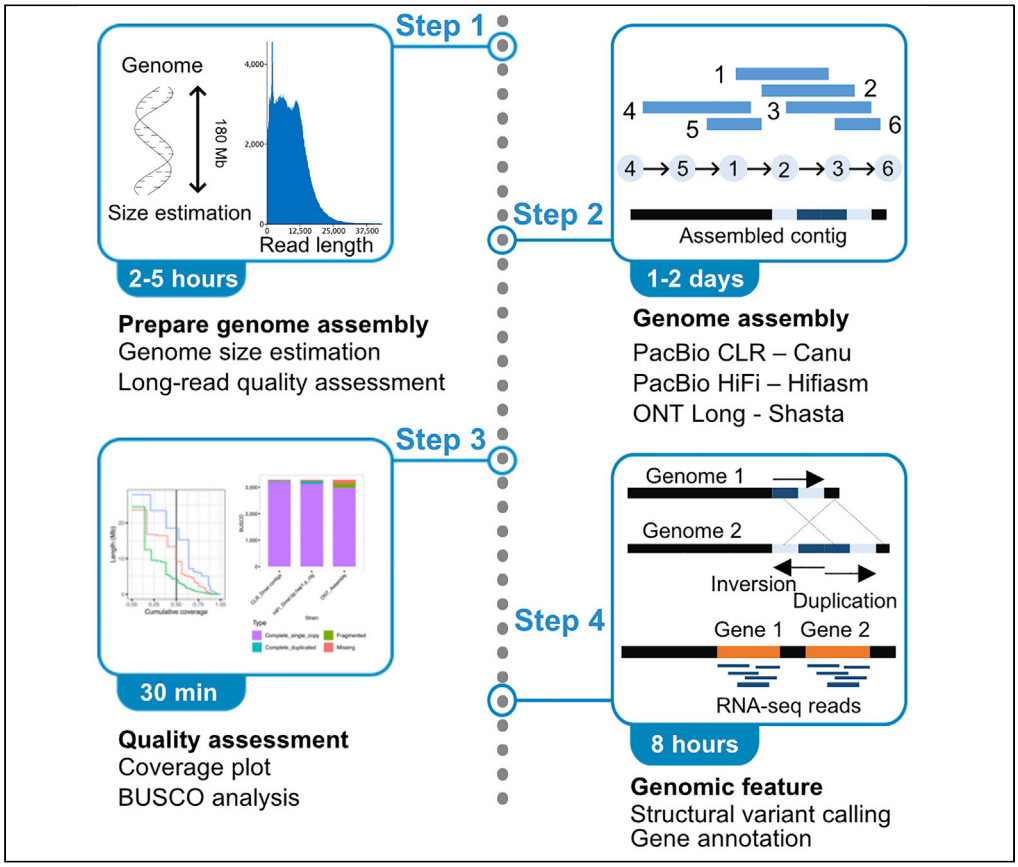
STAR Protocols

CellPress
OPEN ACCESS

Protocol

A beginner's guide to assembling a draft genome and analyzing structural variants with long-read sequencing technologies



Advances in long-read DNA sequencing technologies have enabled researchers to obtain high-quality genomes and finely resolve structural variants (SVs) in many species, even from small laboratories. The hands-on protocol presented here will guide you through the process of analyzing three different types of publicly available *Drosophila melanogaster* datasets obtained using current long-read sequencing technologies. We hope that this protocol will help in guiding researchers who are new to the process of long-read sequencing analysis.

Publisher's note: Undertaking any experimental protocol requires adherence to local institutional guidelines for laboratory safety and ethics.

Jun Kim, Chuna Kim

dauer@snu.ac.kr (J.K.)
kimchuna@kribb.re.kr (C.K.)

Highlights

Hands-on protocol for users who are new to long-read genome assembly

A guide to long-read genome assembly, structural variant calling, and gene annotation

Covers three widely used long-read data types of PacBio and ONT

Analysis and visualization using publicly available *Drosophila melanogaster* data

Kim & Kim, STAR Protocols *3*, 101506
September 16, 2022
https://doi.org/10.1016/j.xpro.2022.101506

STAR Protocols

CellPress
OPEN ACCESS

Protocol

A beginner's guide to assembling a draft genome and analyzing structural variants with long-read sequencing technologies

Jun Kim[1,3,*] and Chuna Kim[2,4,*]

[1]Research Institute of Basic Sciences, Seoul National University, Seoul 08826, Korea

[2]Aging Convergence Research Center, Korea Research Institute of Bioscience and Biotechnology, Daejeon 34141, Korea

[3]Technical contact

[4]Lead contact

*Correspondence: dauer@snu.ac.kr (J.K.), kimchuna@kribb.re.kr (C.K.)
https://doi.org/10.1016/j.xpro.2022.101506

SUMMARY

Advances in long-read DNA sequencing technologies have enabled researchers to obtain high-quality genomes and finely resolve structural variants (SVs) in many species, even from small laboratories. The hands-on protocol presented here will guide you through the process of analyzing three different types of publicly available *Drosophila melanogaster* datasets obtained using current long-read sequencing technologies. We hope that this protocol will help in guiding researchers who are new to the process of long-read sequencing analysis.

BEFORE YOU BEGIN

One of the biggest goals in the genomics field is to obtain the complete genomes and genetic variants of all living organisms. Next-generation sequencing (NGS) technology has made an enormous contribution to our understanding of the relationship between single-nucleotide polymorphisms (SNPs) and various biological phenomena, including cancer, other disease, and evolution. However, variant calling is highly dependent on the quality of the reference genome as it begins with the mapping of NGS reads onto the reference. Furthermore, because of NGS technology's short read lengths (~200 bp), it is difficult to precisely analyze large structural variants (SVs) and genetic variants in repetitive genomic regions, and it remains a challenge to assemble high-quality *de novo* assembled genomes using NGS alone.

Advances in long-read sequencing technology have solved these problems by providing highly accurate (>Q20) or ultra-long (~1 Mb) reads at reasonable costs (Jain et al., 2018; Wenger et al., 2019). Now, using long-read sequencing technology, any genome of any species can be easily assembled, and their SVs can be easily detected. Thus, several consortia, including the Earth BioGenome Project, Darwin Tree of Life Project, and Telomere-to-Telomere (T2T) consortium, have taken this as an opportunity to provide all eukaryotic genomes on Earth or complete the human genome (The Darwin Tree of Life Project Consortium, 2022; Lewin et al., 2018; Nurk et al., 2022). Furthermore, because costs have been dramatically reduced, it is now possible to obtain high-quality *de novo* genome assemblies and SV information even in any laboratories.

Here, we present a step-by-step analysis of long-read DNA sequencing, which includes the software installation, genome assembly, quality assessment, SV calling, and gene annotation (Figure 1). We covered all widely used long-read platforms (Pacific Biosciences continuous long-read and high-fidelity long-read sequencing as well as Oxford Nanopore Technologies long-read sequencing). This hands-on protocol covers the entire process, from public data acquisition to output interpretation;

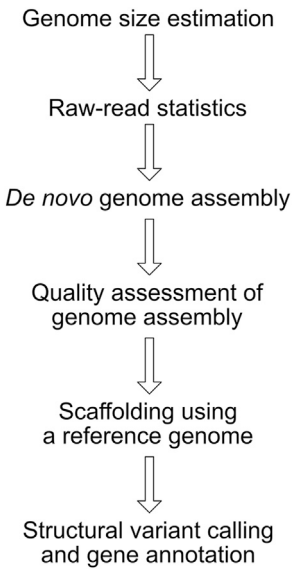


Figure 1. Workflows for analyzing long-read DNA sequencing data

thus, even novice researchers will be able to understand the methodology. We also provided a brief explanation for each step to ensure that as many researchers as possible will be able to understand and apply the steps.

Before you begin, the following are the general conventions used for code chunks: # denotes a non-executable comment; the shebang (#!) specifies whether the script is a bash script or an R script. Both types of script can be saved as a file by copying and pasting them in a text editor, such as vim or nano, and the file can then be run in your terminal with the following command: *bash filename* or *Rscript filename*. If the first line of the code chunk does not contain a shebang and begins with >, the code chunk can be executed directly from your terminal. You should copy and paste the code chunk without the > symbol. The timing presented in this protocol is the time spent using the Linux workstation described in the key resources table. The analysis time may vary depending on the computer environment used and its specifications.

Preparing a conda environment

Timing: 10 min

1. Conda is an open-source environment management system. Miniconda is a minimal installer for Conda. It can run on Windows Subsystem for Linux (WSL), macOS, and Linux.
   a. To download and install Miniconda, go to https://docs.conda.io/en/latest/miniconda.html#latest-miniconda-installer-links. The current pipeline was executed on a high-performance workstation running the Ubuntu operating system (key resources table).
2. In your terminal window, run the following commands sequentially:

```
# Download lastest Miniconda3
> wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
> chmod +x Miniconda3-latest-Linux-x86_64.sh
> bash Miniconda3-latest-Linux-x86_64.sh
```

```
# Press ENTER to read its license, and then enter yes to agree with it
# Specify the path to install your conda
# We used the latest miniconda; python 3.9.5, conda version 4.11.0
# Add the conda to your PATH environment variable
> </path_to_your_conda>/bin/conda init
> source ~/.bashrc
# You can see that your command line has been changed to display ''(base)''
# e.g., (base) username@hostname:~$
# if (base) doesn't appear even after running ''source ~/.bashrc'', then restart your terminal prompt
> conda update -n base -c defaults conda
> conda config -add channels conda-forge
> conda config -add channels bioconda
> conda create -n assembly
> conda activate assembly
# The ''(base)'' should be changed to ''(assembly)''
# e.g., (assembly) username@hostname:~$
# You can deactivate your conda environment using
> conda deactivate
```

Install the required packages in the conda environment

Timing: 10 min

3. Users should install the required packages in the assembly environment (listed in the key resources table). They can be downloaded through Bioconda (https://anaconda.org/bioconda). *conda install* is the command required to install packages.
   a. Install the packages needed for the analysis:

```
# We recommend that you create a conda environment using the specified versions of the following packages to avoid package dependency issues
> conda install -c bioconda kat=2.4.1
> conda install -c bioconda trinity=2.13.2
> conda install -c bioconda assembly-stats bioawk shasta canu hifiasm
> conda install -c bioconda hisat2
> conda install -c conda-forge -c bioconda busco=5.2.2
> conda install -c bioconda ragtag
> conda install -c bioconda svim svim-asm
```

Download the required public datasets

Timing: 1 h

4. When using datasets from public repositories, such as the Sequence Read Archive (SRA) and European Nucleotide Archive (ENA), the download bash scripts can be easily created from SRA explorer (https://sra-explorer.info/) using the accession number of SRA and ENA.
   a. Using the accession number listed in the key resources table, create bash scripts to download the sequence files from the SRA explorer website.
   b. Download the public datasets required for this pipeline using the below Bash scripts:

```
#!/usr/bin/env bash

curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR130/025/SRR13070625/SRR13070625_1.fastq.gz -o SRR13070625_Nanopore_sequencing_of_Drosophila_melanogaster_whole_adult_flies_pooled_male_and_female_1.fastq.gz

curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR124/080/SRR12473480/SRR12473480_subreads.fastq.gz -o SRR12473480_Drosophila_PacBio_HiFi_UltraLow_subreads.fastq.gz

curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR120/022/SRR12099722/SRR12099722_1.fastq.gz -o SRR12099722_WGS_Drosophila_melanogaster_adult_ISCs_1.fastq.gz

curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR120/022/SRR12099722/SRR12099722_2.fastq.gz -o SRR12099722_WGS_Drosophila_melanogaster_adult_ISCs_2.fastq.gz

curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR119/025/SRR11906525/SRR11906525_subreads.fastq.gz -o SRR11906525_WGS_of_drosophila_melanogaster_female_adult_subreads.fastq.gz

curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR151/042/SRR15130842/SRR15130842_1.fastq.gz -o SRR15130842_GSM5452672_Control_CM2_Drosophila_melanogaster_RNA-Seq_1.fastq.gz

curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR151/042/SRR15130842/SRR15130842_2.fastq.gz -o SRR15130842_GSM5452672_Control_CM2_Drosophila_melanogaster_RNA-Seq_2.fastq.gz

curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR151/041/SRR15130841/SRR15130841_1.fastq.gz -o SRR15130841_GSM5452671_Control_CM1_Drosophila_melanogaster_RNA-Seq_1.fastq.gz

curl -L ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR151/041/SRR15130841/SRR15130841_2.fastq.gz -o SRR15130841_GSM5452671_Control_CM1_Drosophila_melanogaster_RNA-Seq_2.fastq.gz

# Download Drosophila melanogaster genome version r6.44 (released Jan 2022)

> wget http://ftp.flybase.net/genomes/Drosophila_melanogaster/dmel_r6.44_FB2022_01/fasta/dmel-all-chromosome-r6.44.fasta.gz
```

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Deposited data		
Drosophila melanogaster reference genome	FlyBase	Genome version: r6.44 (http://ftp.flybase.net/genomes/Drosophila_melanogaster/dmel_r6.44_FB2022_01/fasta/dmel-all-chromosome-r6.44.fasta.gz)
Drosophila melanogaster; Short-read RNA-Seq	NCBI Gene Expression Omnibus	Accession number: GSM5452671 (https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM5452671) Accession number: GSM5452672 (https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM5452672)
Drosophila melanogaster adult ISCs; Short-read whole-genome sequencing	Sequence Read Archive	Accession number: SRX8624462 (https://www.ncbi.nlm.nih.gov/sra/?term=SRX8624462)

(*Continued on next page*)

Continued

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Drosophila melanogaster: female adult; PacBio CLR	Sequence Read Archive	Accession number: SRX8453114 (https://www.ncbi.nlm.nih.gov/sra/?term=SRX8453114)
Drosophila melanogaster F1 females from A4 X ISO1 cross; PacBio HiFi UltraLow	Sequence Read Archive	Accession number: SRX8967562 (https://www.ncbi.nlm.nih.gov/sra/?term=SRX8967562)
Drosophila melanogaster: whole adult flies, pooled male and female; ONT SQK-LSK109+R9.4.1	Sequence Read Archive	Accession number: SRX9518233 (https://www.ncbi.nlm.nih.gov/sra/?term=SRX9518233)
Software and algorithms		
Bioawk v1.0	(Li, 2017)	https://github.com/lh3/bioawk
Assembly-stats v1.0.1	(Wellcome Sanger Institute Pathogen Informatics, 2020)	https://doi.org/10.5281/zenodo.322347
KAT v2.4.1	(Mapleson et al., 2017)	https://github.com/TGAC/KAT
Trinitiy v2.13.2	(Grabherr et al., 2011)	https://github.com/trinityrnaseq/trinityrnaseq
HISAT2 v2.2.1	(Kim et al., 2019b)	https://daehwankimlab.github.io/hisat2/
SAMtools v1.12	(Li et al., 2009)	http://www.htslib.org/
Shasta v0.8.0	(Shafin et al., 2020)	https://github.com/chanzuckerberg/shasta
Canu v2.2	(Koren et al., 2017)	https://github.com/marbl/canu
Hifiasm v0.16.1	(Cheng et al., 2021)	https://github.com/chhylp123/hifiasm
BUSCO v5.2.2	(Manni et al., 2021)	https://busco.ezlab.org/
RagTag v2.1.0	(Alonge et al., 2021)	https://github.com/malonge/RagTag
Minimap2 v2.23	(Li, 2021)	https://github.com/lh3/minimap2
SyRi v1.4	(Goel et al., 2019)	https://schneebergerlab.github.io/syri/
SVIM v1.4.2	(Heller and Vingron, 2019)	https://github.com/eldariont/svim
SVIM-asm v1.0.2	(Heller and Vingron, 2020)	https://github.com/eldariont/svim-asm
RepeatMasker 4.1.0	(Smit et al., 2013–2015)	https://www.repeatmasker.org/
RepeatModeler 2.0.1	(Smit and Hubley, 2008–2015)	https://www.repeatmasker.org/
BRAKER version 2.1.5	(Hoff et al., 2016)	https://github.com/Gaius-Augustus/BRAKER
R software v4.0.5	(R Core Team, 2013)	https://www.r-project.org/
RStudio v1.4.1106	(RStudio Team, 2020)	https://rstudio.com/
ggplot2 package v3.3.5	(Wickham et al., 2016)	https://ggplot2.tidyverse.org/
Tidyverse package v1.3.1	(Wickham et al., 2019)	https://www.tidyverse.org/
Reshape2 package v1.4.4	(Wickham, 2007)	https://cran.r-project.org/web/packages/reshape2/index.html
Dplyr package v1.0.7	(Wickham et al., 2021)	https://dplyr.tidyverse.org/
Cowplot package v1.1.1	(Wilke, 2019)	https://cran.r-project.org/web/packages/cowplot/vignettes/introduction.html
Other		
Hardware: Intel Xeon Gold 6226 processor (12 core), 384-GB RAM, and Ubuntu version 18.04.5	N/A	N/A

STEP-BY-STEP METHOD DETAILS

Visualizing read-length distribution

Timing: 1 h

The continuous long-read (CLR) mode of Pacific Biosciences (PacBio) or Oxford Nanopore Technologies (ONT) will generate reads of varying sizes, thus necessitating the use of statistics to determine whether or not the sequencing was successful. It is important to visualize read-length distributions and ensure that your reads were properly generated. N50, a read- or contig-length distribution statistic, can be used for assessing the read-length quality. N50 is the shortest read or contig length obtained when the cumulative length of the longest read or contig length equals 50% of the total read or assembly length. We present scripts to visualize the read-length distributions of the three long-read datasets.

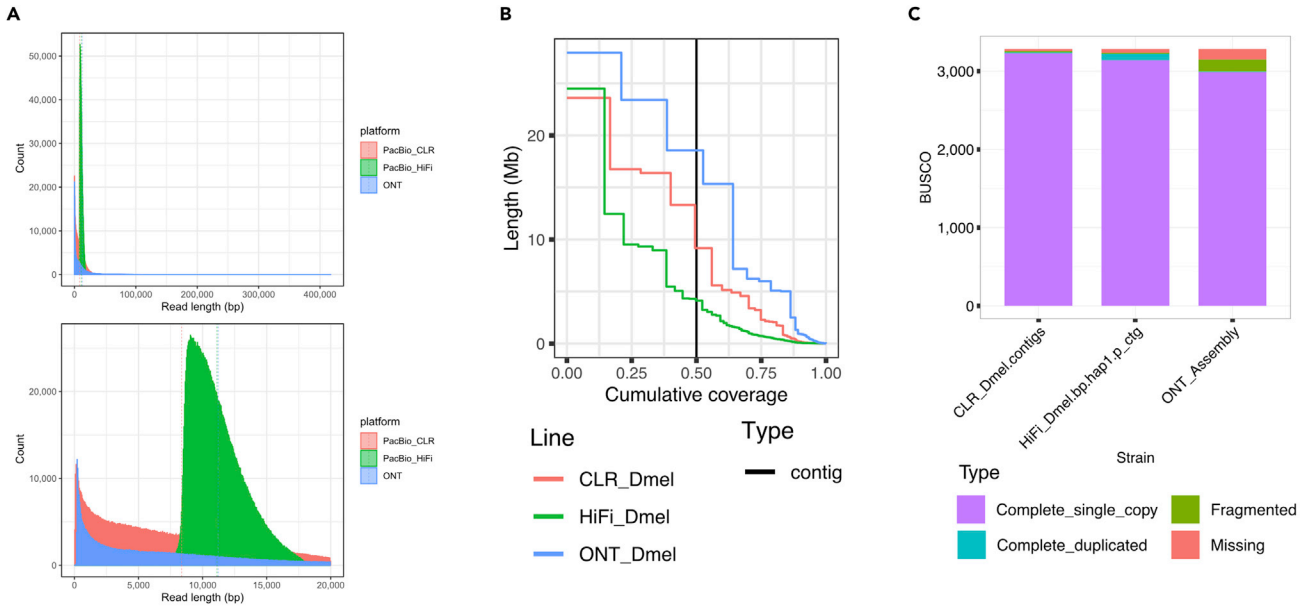


Figure 2. *De novo* genome assembly read-length distribution and quality assessment

(A) Read-length distributions of the three publicly available datasets used in this study. Each vertical dotted line represents the mean value of each dataset.

(B) Cumulative coverage plot for the contig/scaffold length.

(C) BUSCO analysis used to determine the number of single-copy orthologs known in a lineage.

Note: The following scripts contain seven symbols, such as ‘, ’, ", ’, ’, ", and ". These seven symbols appear similar to each other; however, they serve distinct functions in a script. To accurately use the scripts, please do not copy and paste them in MS Word; otherwise, Word may automatically transform one symbol into another, and the script may not function at all.

1. Run the following scripts in your terminal to create a read-length table:

```
#!/usr/bin/env bash

# Create a new file and generate a header line

echo "platform,length" > length.csv

# Add each read length into the length.csv file.

bioawk -c fastx '{print "PacBio_CLR," length($seq)}' SRR11906525_WGS_of_drosophila_melanogaster_female_adult_subreads.fastq.gz >> length.csv

bioawk -c fastx '{print "PacBio_HiFi," length($seq)}' SRR12473480_Drosophila_PacBio_HiFi_UltraLow_subreads.fastq.gz >> length.csv

bioawk -c fastx '{print "ONT," length($seq)}' SRR13070625_1.fastq.gz >> length.csv
```

2. Visualize the read-length distribution data using R ggplot2. Save this script as a new file and run it, or type the following script directly into R or Rstudio. The output will be similar to that presented in Figure 2A:

```
#!/usr/bin/env Rscript

# Please specify your working directory using setwd

setwd("/path/to/Input_CSV_file")

library(ggplot2)

library(dplyr)

library(cowplot)

# Import the read-length distribution table

read_length_df <- read.csv("length.csv")

# Organize the imported read-length table

# You can replace the level arguments for your platform, species, or strains

read_length_df$platform <- as.factor(read_length_df$platform)

read_length_df$platform <- factor(read_length_df$platform,level = c("PacBio_CLR",
"PacBio_HiFi","ONT"))

# Calculate the average read-lengths for each platform

summary_df <- ddply(read_length_df, "platform", summarise, grp.mean=mean(length))

# Draw a read-length distribution plot for all reads

total.length.plot <- ggplot(read_length_df, aes(x=length, fill=platform, color=plat
form)) +

  geom_histogram(binwidth=100, alpha=0.5, position="dodge") +

  geom_vline(data=summary_df, aes(xintercept=grp.mean, color=platform), linetype="-
dashed", size =0.2) +

  scale_x_continuous(labels = comma) +

  scale_y_continuous(labels = comma) +

  labs(x = "Read length (bp)", y = "Count") +

  theme_bw()

# Draw a read-length distribution plot for reads ≤ 20 kb in length

20 kb.length.plot <- ggplot(read_length_df, aes(x=length, fill=platform, color=platform)) +

  geom_histogram(binwidth=50, alpha=0.5, position="dodge") +

  geom_vline(data=summary_df, aes(xintercept=grp.mean, color=platform), linetype=
"dashed", size=0.2) +

  scale_x_continuous(labels = comma, limit = c(0,20000)) +

  scale_y_continuous(labels = comma) +

  labs(x = "Read length (bp)", y = "Count") +

  theme_bw()

# Merge both the read-length distribution plots

plot <- plot_grid(total.length.plot, 20 kb.length.plot, ncol = 1)

# Save the figure using the file name, ''read.length.pdf''

pdf("read.length.pdf",width=6,height=8,paper='special')

print(plot)

dev.off()
```

3. Calculate N50 statistics using assembly-stats. You can save or type this script in your terminal to run it:

```
#!/usr/bin/env bash

# Unzipped FASTA/Q files are required for assembly-stats

# You can unzip your fastq.gz files using the command ''gzip -d file_name.fastq.gz''

# For general usage, specify the read or contig file names after ''assembly-stats''

# Calculate summary stats and save the output as an ''N50_stat'' file

assembly-stats  SRR11906525_WGS_of_drosophila_melanogaster_female_adult_subreads.fastq
>> N50_stat

assembly-stats SRR12473480_Drosophila_PacBio_HiFi_UltraLow_subreads.fastq >> N50_stat

assembly-stats SRR13070625_1.fastq >> N50_stat
```

```
# You can see the output of assembly-stats by typing ''cat N50_stat'' in your terminal

> cat N50_stat

# The following is the output of ''cat N50_stat'' command (result of assembly-stats)

stats for SRR11906525_WGS_of_drosophila_melanogaster_female_adult_subreads.fastq

sum = 12016661679, n = 1437524, ave = 8359.28, largest = 99345

N50 = 13094, n = 321336

N60 = 11342, n = 419876

N70 = 9489, n = 535376

N80 = 7388, n = 678061

N90 = 4902, n = 874814

N100 = 50, n = 1437524

N_count = 0

Gaps = 0

stats for SRR12473480_Drosophila_PacBio_HiFi_UltraLow_subreads.fastq

sum = 25600110705, n = 2301518, ave = 11123.14, largest = 26462

N50 = 11151, n = 976954

N60 = 10586, n = 1212627

N70 = 10055, n = 1460775

N80 = 9530, n = 1722273

N90 = 8996, n = 1998694

N100 = 369, n = 2301518

N_count = 0

Gaps = 0

stats for SRR13070625_1.fastq
```

```
sum = 7133020037, n = 640215, ave = 11141.60, largest = 417450

N50 = 21491, n = 83878

N60 = 16642, n = 121685

N70 = 12824, n = 170598

N80 = 9526, n = 235039

N90 = 6112, n = 327186

N100 = 1, n = 640215

N_count = 0

Gaps = 0
```

Note: For the PacBio CLR mode and ONT, high-quality DNA would have >10-kb N50 read lengths, and a high-quality genome assembly would have >1-Mb N50 contig lengths (Kim et al., 2019a, 2020, 2021).

Approximate genome-size estimation

Timing: 5 h

This part of the protocol is required when generating data for a novel species. After estimating the genome size, you can determine the required sequencing throughput for your species. A high-quality genome assembly necessitates more than 20× sequencing coverage. We propose three methodologies that can be employed depending on the situation. You can skip this step if you are analyzing public datasets.

4. The estimated genome size of your species can be found in public databases:
   a. Animal: Animal Genome Size Database (http://www.genomesize.com/index.php)
   b. Plant: Plant DNA C-values Database (https://cvalues.science.kew.org/
5. If you have short-read DNA sequencing data, the k-mer-based genome size estimation can be applied:

```
#!/usr/bin/env bash

# KAT is a toolkit for addressing assembly completeness through k-mer counts (Mapleson et al., 2017)

# More information about KAT in: https://github.com/TGAC/KAT

# You can use the short-read DNA sequencing data provided in the Key Resource Table (Accession number: SRX8624462) to run the following script

# You need to provide the file path to the sequencing data or run this script in the same folder where the sequencing data is saved

kat hist -o prefix -t 10 SRR12099722* 1> kat.output.txt

echo dme_size >> genome_size.txt

grep -i "Estimated" kat.output.txt >> genome_size.txt

# hist: a kat module for drawing histograms and estimating genome size

# -o: output prefix; you can specify ''prefix'' for your species or strain names
```

```
# -t: the number of threads that will be used to run the kat program

# You can replace SRR12099722* with your short-read DNA sequencing data# You can replace dme_-
size with the name of your species
```

```
# You can check the kat output by typing ''cat genome_size.txt'' in your terminal

> cat genome_size.txt

# Genome size can be estimated using the short-read DNA sequencing data

dme_size

Estimated genome size: 166.18 Mbp

Estimated heterozygous rate: 0.41%
```

6. Calculate the transcript-based coverage using short-read DNA/RNA sequencing data.

⚠ CRITICAL: For an accurate estimation, high-quality transcriptome assembly is required.

a. Conduct *de novo* transcriptome assembly using Trinity (Grabherr et al., 2011):

```
#!/usr/bin/env bash

# Trinitiy is a package for conducting de novo transcriptome assembly from RNA-seq data

# For more information: https://github.com/trinityrnaseq/trinityrnaseq/wiki

# You can use the short-read RNA sequencing data provided in the Key Resource Table (Accession
number: GSM5452671, GSM5452672) to run the following script

# You need to provide the file path to the sequencing data or run this script in the same folder
where the sequencing data are saved

Trinity -seqType fq -max_memory 120G -left /home/assembly/analysis/00_STARprotocol/
SRR15130841_GSM5452671_Control_CM1_Drosophila_melanogaster_RNA-Seq_1.fastq.gz,/home/
assembly/analysis/00_STARprotocol/SRR15130842_GSM5452672_Control_CM2_Drosophila_
melanogaster_RNA-Seq_1.fastq.gz -right /home/assembly/analysis/00_STARprotocol/
SRR15130841_GSM5452671_Control_CM1_Drosophila_melanogaster_RNA-Seq_2.fastq.gz,/home/
assembly/analysis/00_STARprotocol/SRR15130842_GSM5452672_Control_CM2_Drosophila_mela-
nogaster_RNA-Seq_2.fastq.gz -CPU 8 -output Dmel.trinity

# -seqType: sequence type; as short-read sequencing data are typically present in the FASTQ
format, you can specify this as ''fq''

# # -max_memory: maximum memory required to run the Trinity. ''120G'' indicates 120 GB

# -left and -right: input files required for trinity analysis. Currently, short-read
sequencing is mainly performed in a ''paired-end'' mode. Each DNA molecule is sequenced at
both the ends, producing two paired files. You should specify one as ''-left'' and the other as
''-right''

# -CPU: the number of threads required for Trinity analysis

# -output: output prefix

# Trinity output should be in the ''Dmel.trinity'' (or ''your_species_trinity'') folder

# Assembled transcript FASTA file will be ''Dmel.trinity.Trinity.fasta'' (or
''your_species_trinity.Trinity.fasta'')
```

```
# You can assess the assembled quality of transcriptomes using assembly-stat

> assembly-stats Dmel.trinity.Trinity.fasta

# The following is the output of the ''assembly-stats Dmel.trinity.Trinity.fasta'' command
(result of assembly-stats)

stats for Dmel.trinity.Trinity.fasta

sum = 72662995, n = 67038, ave = 1083.91, largest = 27780

N50 = 2454, n = 8357

N60 = 1816, n = 11781

N70 = 1180, n = 16695

N80 = 653, n = 25022

N90 = 364, n = 40185

N100 = 201, n = 67038

N_count = 0

Gaps = 0
```

b. Map the short DNA reads to the transcriptome using HISAT2:

```
#!/usr/bin/env bash

# HISAT2 was used to map DNA sequencing reads to the assembled transcripts, and SAMtools was
used to process the alignment data

# For more information about HISAT2: http://daehwankimlab.github.io/hisat2/

# HISAT2 ref: https://www.nature.com/articles/s41587-019-0201-4

# For more information about SAMtools: http://www.htslib.org/

# SAMtools ref: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2723002/

# Index your assembled transcript FASTA file using the prefix ''Dmel''

hisat2-build Dmel.trinity.fa Dmel

# Map your short-read DNA sequences to the assembled transcript using the index

# You can use the short-read DNA sequencing data provided in the Key Resource Table (Accession
number: SRX8624462) to run the following script

# You need to provide the file path to the sequencing data or run this script in the same folder
where the sequencing data are saved

hisat2 -x Dmel -p 10 -1 SRR12099722*_1* -2 SRR12099722*_2* –very-sensitive | samtools sort -@
10 -o Dmel.very_sensitive.bam

# For HISAT2, the parameters are as follows:

# -x: index prefix

# -p: the number of threads required by HISAT2

# -1 and -2: paired-end files; you can change the name of your sequencing data

# –very-sensitive: sensitivity option

# For SAMtools, the parameters are as follows:
```

```
# sort: SAMtools module to sort the mapped read information

# -@: the number of threads required by SAMtools

# -o: output file name

# Index your read mapping file

samtools index Dmel.very_sensitive.bam
```

c. Estimate the genome size:

```
#!/usr/bin/env bash

# Calculate average coverage of each transcript

samtools coverage Dmel.very_sensitive.bam | awk '{print $7}' | tail -n +2 | grep -vw "0" | awk '{sum+=$1}END{print sum/NR}' > average.coverage.txt

# coverage: SAMtools module to calculate the coverage of each transcript (or contig, scaffold, etc.)

# awk '{print $7}': select the coverage column in the output of SAMtools coverage

# tail -n +2: remove the header line

# grep -vw "0": remove ''0'' coverage rows

# awk '{sum+=$1}END{print sum/NR}': calculate the average coverage with non-zero values

# Calculate the total read length of the DNA sequencing file

bioawk -c fastx '{sum+=length($seq)}END{print sum}' SRR12099722_WGS_Drosophila_melanogaster_adult_ISCs_1.fastq.gz > total.read.length.txt

# Estimate the genome size

paste average.coverage.txt total.read.length.txt | awk '{print "Estimated genome size = " $2*2/$1/1000000 " Mb"}'
```

```
# After running the preceding script, the following result will be displayed in your terminal

Estimated genome size = 185.04 Mb
```

Long-read sequencing-based genome assembly

Timing: 1 day for step 7

Timing: 1.5 day for step 8

Timing: 30 min for step 9

Long-read sequencing data are now typically produced using PacBio or the ONT sequencing technology. Here, we summarize the assembly method when using PacBio's two data types, i.e., CLR and high-fidelity (HiFi) modes, as well as ONT's Long data type. You can select one of the 7–9 scripts according to your data type:

7. PacBio CLR data type:

```
# Typically, canu assembler (Koren et al., 2017) will use as much as CPU and memory resources in your computer

# You can use the PacBio CLR data provided in the Key Resource Table (Accession number: SRX8453114) to run the following command

> canu -p Dmel -d Dmel genomeSize=170 m -pacbio SRR11906525_WGS_of_drosophila_melanogaster_female_adult_subreads.fastq.gz

# -p: output prefix

# -d: directory where Canu will run

# genomeSize=: estimated genome size of your species

# -pacbio: name of your platform

# For more information about Canu: https://github.com/marbl/canu
```

```
# You can check the assembly statistics of the canu assembler using assembly-stats

> assembly-stats Dmel.contigs.fasta

# After running the preceding command, the assembly statistics of the Canu assembler will be displayed on your terminal

stats for Dmel.contigs.fasta

sum = 141740149, n = 452, ave = 313584.40, largest = 23607911

N50 = 9177974, n = 5

N60 = 5147831, n = 7

N70 = 4576628, n = 9

N80 = 2051575, n = 15

N90 = 187381, n = 39

N100 = 1381, n = 452

N_count = 0

Gaps = 0
```

8. PacBio HiFi data type:
   a. Construct a genome using the hifiasm assembler (Cheng et al., 2021), which is dedicated to

```
# You can use the PacBio HiFi data provided in Key Resource Table (Accession number: SRX8967562) to run the following command

> hifiasm -o Dmel -t 20 ../SRR12473480_Drosophila_PacBio_HiFi_UltraLow_subreads.fastq.gz

# -o: output prefix

# -t: the number of threads

# For more information about hifiasm: https://github.com/chhylp123/hifiasm
```

   the HiFi data type:
   b. Convert the GFA file to a typical FASTA file:

```
# You can check assembly statistics of the hifiasm assembler using assembly-stats

> assembly-stats Dmel.bp.hap1.p_ctg.fa

# After running the preceding command, the assembly statistics of the hifiasm assembler will be displayed on your terminal

stats for Dmel.bp.hap1.p_ctg.fa

sum = 168692738, n = 654, ave = 257939.97, largest = 24502687

N50 = 4127200, n = 10

N60 = 2167675, n = 15

N70 = 1125434, n = 26

N80 = 496545, n = 50

N90 = 79880, n = 130

N100 = 9867, n = 654

N_count = 0

Gaps = 0
```

Note: The HiFi sequencing data used in this guide were generated using ultra-low input DNA; thus, these data significantly differ from typical HiFi data with sufficient input DNA. PacBio HiFi data are typically generated through a strict size selection, with an average quality > Q30. The HiFi read-length distribution will be 15–20 kb, and HiFi data for diploid genome assembly are typically superior to CLR data in terms of phasing, contiguity, and computation time. Because of the high accuracy of the process, diploid variants can be resolved and phased more easily, and the correction step required for CLR data can be omitted for HiFi data.

9. ONT Long data type:

```
# Shasta (Shafin et al., 2020) is a long-read sequencing assembler which works efficiently on ONT data. Raw-read FASTQ files should be unzipped for Shasta assembler

# You can use the ONT Long data provided in Key Resource Table (Accession number: SRX9518233) to run the following command

# Unzip your ONT raw-read FASTQ file

> gzip -d SRR13070625_1.fastq.gz

# Run shasta to assemble the reads into contigs

> shasta –config Nanopore-Oct2021 –threads 8 –input SRR13070625_1.fastq

# –config: configuration options

# –threads: the number of threads required by Shasta

# –input: input file name; the file should be unzipped

# For more information about Shasta: https://github.com/chanzuckerberg/shasta

# You can check assembly statistics of the shasta assembler using assembly-stats

> assembly-stats Assembly.fasta

# After running the preceding command, assembly statistics of the shasta assembler will be displayed on your terminalstats for Assembly.fasta

sum = 133002022, n = 208, ave = 639432.80, largest = 27938801
```

```
N50 = 18567724, n = 3

N60 = 15335596, n = 4

N70 = 6235146, n = 6

N80 = 5092624, n = 8

N90 = 917306, n = 13

N100 = 21, n = 208

N_count = 0

Gaps = 0
```

Quality assessment

Timing: 10 min for step 10

Timing: 20 min/sample for step 11

Timing: 10 min/sample for step 12

The quality of a *de novo* assembled genome can be determined according to the contiguity of its contigs, which can be determined by the length of contigs and identification of universal single-copy ortholog genes. Furthermore, if a high-quality reference genome exists for the species you have assembled, the quality can be evaluated using a comparison to your own genome.

10. Produce a coverage plot.

This cumulative coverage plot depicts contig-length distributions. Contig lengths are sorted in descending order, and the proportion of each contig length to its total genome assembly length is calculated. Their cumulative sum is shown on the x-axis, and the length of the corresponding contig is presented on the y-axis. Based on the definition of N50, each horizontal line that crosses the vertical line in each assembly can be interpreted as N50, which allows different assemblies to be visually compared.

a. Conduct preprocessing:

```
#!/usr/bin/env bash

# This script will create the coverage table required to obtain the cumulative graph

STRAIN1=Hifi_Dmel # Specify your species or strain name

REF1=/path/to/Hifi_Dmel.bp.hap1.p_ctg.fa # should be changed for your genome file path

TYPE1=contig # Specify your genome assembly type, such as contig, scaffold, chromosome, etc.

LEN1=`bioawk -c fastx '{sum+=length($seq)}END{print sum}' $REF1` # Size of assembled genome

# Create the output file having a header line

echo "line,length,type,coverage" > length.csv

# Calculate cumulative sum and write result to the output file (HiFi data)

cat $REF1 | bioawk -c fastx -v line="$STRAIN1" '{print line","length($seq)","length($seq)}'
| sort -k3rV -t "," | awk -F "," -v len="$LEN1" -v type="$TYPE1" 'OFS=","{ print $1,$2,
type,(sum+0)/len; sum+=$3 }' >> length.csv
```

```
# Calculate cumulative sum and write result to the output file (CLR data)

STRAIN2=CLR_Dmel

REF2=/path/to/CLR_Dmel.contigs.fasta # should be changed your genome name

TYPE2=contig

LEN2=`bioawk -c fastx '{sum+=length($seq)}END{print sum}' $REF2`

cat $REF2 | bioawk -c fastx -v line="$STRAIN2" '{print line","length($seq)","length($seq)}'
| sort -k3rV -t "," | awk -F "," -v len="$LEN2" -v type="$TYPE2" 'OFS=","{ print
$1,$2,type,(sum+0)/len; sum+=$3 }' >> length.csv

# Calculate cumulative sum and write result to the output file (ONT data)

STRAIN3=ONT_Dmel

REF3=/path/to/ONT_Assembly.fasta # should be changed your genome name

TYPE3=contig

LEN3=`bioawk -c fastx '{sum+=length($seq)}END{print sum}' $REF3`

cat $REF3 | bioawk -c fastx -v line="$STRAIN3" '{print line","length($seq)","length($seq)}'
| sort -k3rV -t "," | awk -F "," -v len="$LEN3" -v type="$TYPE3" 'OFS=","{ print
$1,$2,type,(sum+0)/len; sum+=$3 }' >> length.csv
```

b. Make a cumulative graph. Save this script as a new file and run it, or type the following script directly into R or RStudio. The output will be similar to that presented in Figure 2B:

```
#!/usr/bin/env Rscript

setwd("/path/to/Input_CSV_file")

library(ggplot2)

# Import the cumulative sum table

contig_cumulative_sum_df <- read.csv("length.csv", header = TRUE)

# Organize the table

contig_cumulative_sum_df$type <- factor(contig_cumulative_sum_df$type, levels=c("scaf-
fold", "contig")) # or any other assembly types

# Create a plot for cumulative sum

plot <- ggplot(data=contig_cumulative_sum_df, aes(x=coverage, y=length/1000000,
color=line)) +

  geom_vline(xintercept = 0.5, linetype="dotted", size=0.5) +

  xlim(0, 1) +

  geom_step(aes(linetype=type)) +

  labs(x = "Cumulative coverage", y = "Length (Mb)")

# Save the plot as a ''coverage.pdf'' file

pdf("coverage.pdf",width=4,height=3,paper='special')

print(plot)

dev.off()
```

11. Perform Benchmarking Universal Single-Copy Orthologs (BUSCO) analysis (Manni et al., 2021). BUSCO analysis determines whether well-known single-copy orthologs in specific lineages are correctly assembled or fragmented in contigs of a genome assembly. In a more contiguous genome assembly, complete BUSCO values would be higher.

a. Select the specific lineage of your species among the following datasets:

```
> busco –list-datasets

# For more information about BUSCO: https://busco.ezlab.org/
```

b. Run the BUSCO analysis:

```
#!/usr/bin/env bash

for assembly in `ls ../*fasta*`;do

name=$(basename -s .fasta $assembly)

busco -i $assembly -c 10 -o $name -m genome -l diptera_odb10

done

# -c 10: number of threads to run BUSCO

# -m genome: mode of BUSCO

# -l diptera_odb10: lineage-specific dataset name selected in the list generated by the ''busco -list-datasets'' command

# For general usage, use this script:

# > busco -i assembly.fasta -o species_name -m genome -l your_lineage
```

c. Parse the BUSCO output results:

```
#!/usr/bin/env bash

# BUSCO will measure the quality of single copy orthologs in four different categories: ''complete and single-copy,'' ''complete and duplicated,'' ''fragmented,'' and ''missing.'' This script will parse the number of data points in each of the categories to create a boxplot

# Create the BUSCO output file having a header line

echo "Strain,Complete_single_copy,Complete_duplicated,Fragmented,Missing" > busco.csv

# Extract the count for each BUSCO category (CLR data)

PREFIX1=CLR_Dmel.contigs

# (S) represents ''complete and single-copy''

cat $PREFIX1/short*.txt | grep "(S)" | awk -v strain="$PREFIX1" '{print strain", "$1}' > complete_single.txt

# (D) represents complete and duplicated

cat $PREFIX1/short*.txt | grep "(D)" | awk '{print $1}' > complete_duplicated.txt

# (F) represents ''fragmented''

cat $PREFIX1/short*.txt | grep "(F)" | awk '{print $1}' > fragmented.txt

# (M) represents ''missing''

cat $PREFIX1/short*.txt | grep "(M)" | awk '{print $1}' > missing.txt
```

```
paste -d "," complete_single.txt complete_duplicated.txt fragmented.txt missing.txt >> busco.csv

# Extract the count for each BUSCO category (HiFi data)

PREFIX2=Hifi_Dmel.bp.hap1.p_ctg

cat $PREFIX2/short*.txt | grep "(S)" | awk -v strain="$PREFIX2" '{print strain", "$1}' > complete_single.txt

cat $PREFIX2/short*.txt | grep "(D)" | awk '{print $1}' > complete_duplicated.txt

cat $PREFIX2/short*.txt | grep "(F)" | awk '{print $1}' > fragmented.txt

cat $PREFIX2/short*.txt | grep "(M)" | awk '{print $1}' > missing.txt

paste -d "," complete_single.txt complete_duplicated.txt fragmented.txt missing.txt >> busco.csv

# Extract the count for each BUSCO category (ONT data)

PREFIX3=ONT_Assembly

cat $PREFIX3/short*.txt | grep "(S)" | awk -v strain="$PREFIX3" '{print strain", "$1}' > complete_single.txt

cat $PREFIX3/short*.txt | grep "(D)" | awk '{print $1}' > complete_duplicated.txt

cat $PREFIX3/short*.txt | grep "(F)" | awk '{print $1}' > fragmented.txt

cat $PREFIX3/short*.txt | grep "(M)" | awk '{print $1}' > missing.txt

paste -d "," complete_single.txt complete_duplicated.txt fragmented.txt missing.txt >> busco.csv

# Delete temporary files

rm complete_single.txt complete_duplicated.txt fragmented.txt missing.txt

# You can check the table summarizing BUSCO output in your terminal

> cat busco.csv

# After running the preceding command, BUSCO result will be displayed in your terminal

Strain,Complete_single_copy,Complete_duplicated,Fragmented,Missing # Header

CLR_Dmel.contigs,3228,13,18,26

Hifi_Dmel.bp.hap1.p_ctg,3140,81,16,48

ONT_Assembly,2989,7,153,136
```

d. Visualize the results using the following R script:

```
#!/usr/bin/env Rscript

setwd("/path/to/Input_CSV_file")

library(ggplot2)

library(reshape2)

library(tidyverse)

# Import the BUSCO table

busco_df <- read.csv("busco.csv", header = TRUE)
```

```
# Organize and rearrange the imported table

busco_df$Strain <- as.factor(busco_df$Strain)

busco_df.melted <- melt(busco_df, id.vars = "Strain")

busco_df.melted$variable <-relevel(busco_df.melted$variable, "Missing")

# Create a stacked bar plot for the BUSCO outputs

busco_plot <- ggplot(busco_df.melted, aes(x=Strain, fill=fct_rev(variable), y=value)) +

  geom_bar(position= "stack", width = 0.7, stat="identity") +

  labs(x = "Strain", y = "BUSCO", fill = "Type") +

  scale_y_continuous(labels=comma) +

  theme_bw() +

  theme(axis.text.x = element_text(angle=45, hjust=1, size = 12), axis.text.y = element_-
text(size = 12), axis.title=element_text(size=12))

# Save the plot as ''busco.pdf''

pdf("busco.pdf",width=8,height=5,paper='special')

print(busco_plot)

dev.off()
```

12. Compare your genome with the reference genome. This step is highly recommended if a chromosome-level reference genome is available.

If a chromosome-level genome assembly is already available, you can connect your contigs into larger chunks using homology between your contigs and the chromosomes. Such larger chunks with unidentified gaps are referred to as "scaffolds."

a. Make the scaffolds using RagTag (Alonge et al., 2021):

```
#!/usr/bin/env bash

for assembly in `ls ../*fasta*`;do

ref=/path/to/reference/dmel-all-chromosome-r6.44.fasta

name=$(basename -s .fasta $assembly)

ragtag.py scaffold -t 10 -u -o $name $ref $assembly

done

# -t: the number of threads required by RagTag

# -u: add a suffix to all unscaffolded contigs

# -o: output folder name

# Final output scaffolds should be saved in ''$name/ragtag.scaffold.fasta''

# For general usage, you can use this script:

# > ragtag.py scaffold -u -o output_folder_name reference.fasta your_assembly.fasta

# For more information about RagTag: https://github.com/malonge/RagTag
```

b. Prepare genomic FASTA files, which have common chromosomes, to compare synteny between a chromosome-level reference genome (reference genome) and your scaffolds (RagTag output):

```
#!/usr/bin/env bash

#1. Remove RagTag identifier from the header of scaffold

for scaffold in `ls ../ragtag.*`;do

name=$(basename -s .scaffold.fasta $scaffold)

sed 's/_RagTag//' $scaffold > ${name}_rename.scaffold.fasta

done

#2. Only chromosomes with the same name should be left in both genomic FASTA files

# chromosome.name.list.txt: The names of the chromosomes to be compared are contained in this file

for i in `cat chromosome.name.list.txt`; do

  cat chromosome-level_genome_assembly.fa | bioawk -c fastx -v chr="$i" '$name==chr{print ">chr"$name; print $seq}' >> reference_chromosome.fa

  cat ragtag.scaffold.fasta | bioawk -c fastx -v chr="$i" '$name==chr{print ">chr"$name; print $seq}' >> your_scaffold.fa

done
```

```
# To run the preceding script, the chromosome.name.list.txt file should be provided

# Example of chromosome name list file contain main chromosomes of Drosophila melanogaster

> cat chromosome.name.list.txt''

# Standard output of ''cat chromosome.name.list.txt''

# By copying and pasting the result below, you can create chromosome.name.list.txt file

2L

2R

3L

3R

4

X

Y
```

c. Perform whole-genome alignment using minimap2 (Li, 2021):

```
> minimap2 -a -x asm5 -eqx reference_chromosome.fa your_scaffold.fa > syri.sam

# -a: output will be saved as the SAM format

# -x asm5: preset for aligning two assemblies with ~0.1% sequence divergence

# -eqx: contain =/X CIGAR strings

# For more information about minimap2: https://github.com/lh3/minimap2
```

d. Make a conda environment for synteny analysis using SyRi (Goel et al., 2019):

```
# At the time of writing, SyRi only functions in Python 3.5, so you should specify the Python version that conda will employ
> conda create -n syri python=3.5
# Activate the environment for SyRi
> conda activate syri
# Install dependencies for SyRi
> conda install cython numpy scipy pandas=0.23.4 biopython psutil matplotlib=3.0.0
> conda install -c conda-forge python-igraph
> conda install -c bioconda pysam
# Then download SyRi version 1.4 and unzip the downloaded file
> wget https://github.com/schneebergerlab/syri/archive/refs/tags/v1.4.tar.gz
> tar -xzf v1.4.tar.gz
> cd syri-1.4
# Install SyRi
> python setup.py install
# Let the SyRi command executable
> chmod +x syri/bin/syri
# For more information about SyRi: https://schneebergerlab.github.io/syri/
```

Note: Currently, SyRi only works with Python 3.5 version.

e. Run SyRi to visualize the synteny information:

```
#!/usr/bin/env bash
#1. Run SyRi
python /path/to/syri-1.4/syri/bin/syri -c syri.sam -r chromosome-level_genome_assembly.fa -q your_scaffold.fa -k -F S
# -k: keep intermediate files; you can turn off this option
# -F S: input file is in the SAM (S) format
#2. Visualizing genomic alignments predicted by SyRi
python /path/to/syri-1.4/syri/bin/plotsr syri.out chromosome-level_genome_assembly.fa your_scaffold.fa -H 8
# -H: Specify the height of the plot
```

Discovery of structural variation

Timing: 10 min/sample

Structural variations (SVs) are genetic variants that differ in size by ≥50 bp from the reference genome. Long-read sequencing technologies out-perform short-read sequencing ones in terms of SV accuracy and specificity owing to their larger read size.

Two methods are available for calling SVs: read-based SV calling and assembly-based SV calling. For read-based SV calling, you should map your long reads to a reference genome before calling SVs using the mapping information. For assembly-based SV calling, you should align your genome assembly to a reference genome before calling SVs. Assembly-based SV calling is typically more accurate than read-based SV calling because most of the read errors are corrected during genome assembly; however, it requires significantly greater sequencing read depth because *de novo* genome assembly requires ~20× coverage.

SVIM (Heller and Vingron, 2019) and SVIM-asm (Heller and Vingron, 2020) are sister SV callers developed for read- and assembly-based SV calling, respectively. Both SV callers are simple to install and easy to run. If you have low-depth read data, use SVIM; if you have high-depth read data and the corresponding genome assembly, use SVIM-asm. Smaller variants can be determined by both SVIM and SVIM-asm using the "–min sv size" option; for example, "–min sv size 5" to call ≥5-bp variants.

13. Modify the SVIM and SVIM-asm figure output options:

```
# First, you should find your path to SVIM_plot.py for SVIM

> whereis svim | sed 's\/bin/svim\/lib/python3.*/site-packages/svim/SVIM_plot.py\; s\svim: \\'

# Example result of the above code: /home/assembly/miniconda3/envs/assembly/lib/python3.*/site-packages/svim/SVIM_plot.py

# for SVIM-asm

> whereis svim-asm | sed 's\/bin/svim-asm\/lib/python3.*/site-packages/svim_asm/SVIM_plot.py\; s\svim-asm: \\'

# Example result of the above code: /home/assembly/miniconda3/envs/assembly/lib/python3.*/site-packages/svim_asm/SVIM_plot.py

# check the printed path and replace "png" to "pdf"

> sed -i 's/png/pdf/' /path/to/envs/env_name/lib/python3.*/site-packages/svim/SVIM_plot.py # for SVIM

> sed -i 's/png/pdf/' /path/to/envs/env_name/lib/python3.*/site-packages/svim_asm/SVIM_plot.py # for SVIM-asm

# Then run your SVIM or SVIM-asm
```

14. Conduct read-based SV calling using SVIM:

```
> svim reads -cores 10 -aligner minimap2 output_folder_name your_read.fq.gz your_genome_assembly.fa

# reads: SVIM module for detecting SVs using raw reads rather than SAM/BAM alignment files

# -cores 10: number of threads

# -aligner: You can use other long-read aligners by changing ''minimap2'' to your desired aligner

# your_read.fq.gz: should be long-read sequencing data

# For more information about SVIM: https://github.com/eldariont/svim
```

15. Conduct assembly-based SV calling.
    a. Align two genomes using minimap2:

```
# Align your genome assembly to the reference genome and sort the alignment information
> minimap2 -a -x asm5 -cs -r2k -t 10 genome1.fa genome2.fa | samtools sort -m4G -@ 10 -O BAM -o genome2_to_genome1.bam # genome1=reference, genome2=query
# For minimap2, the parameters are as follows:
# -a: output will be printed as the SAM format
# -x asm5: preset for aligning two assemblies with ~0.1% sequence divergence
# -cs: the output file will contain cs tags
# -r: chaining bandwidth
# -t: number of threads
# For SAMtools, the parameters are as follows:
# sort: SAMtools module to sort read mapping information
# -m: maximum memory for each thread
# -@: number of threads
# -O BAM: output as a BAM format
# Index your assembly-assembly alignment file
> samtools index genome2_to_genome1.bam
```

    b. Perform SV calling using SVIM-asm:

```
# Call SVs between the reference genome and yours
> svim-asm haploid output_folder_name genome2_to_genome1.bam genome1.fa
# haploid: SVIM-asm module for calling SVs between two haploid genomes
# For more information about SVIM-asm: https://github.com/eldariont/svim-asm
```

Gene annotation

Timing: 8 h/sample

To annotate genes for your genome, you should (1) mask your genome assembly, (2) map your RNA-seq reads to the masked genome assembly, and (3) predict gene structures based on this RNA-seq evidence. The BRAKER gene annotation pipeline, which will be used by us, prefers repeat-masked genome assemblies to unmasked ones to accurately determine the gene structure (Hoff et al., 2016). The repeat-masking process can be performed using RepeatMasker (Smit et al., 2013–2015) and RepeatModeler (Smit and Hubley, 2008–2015). Additionally, as coding and non-coding genes are transcribed to produce RNA molecules, RNA-seq data provide important evidence for gene structure.

16. Make a conda environment for repeat masking:

```
# Type the following script directly in your terminal

# Create the conda environment for RepeatModeler and RepeatMasker

# The RepeatModeler package contains the RepeatMasker package

> conda create -c bioconda -n repeatmodeler repeatmodeler

# Activate the environment for RepeatModeler

> conda activate repeatmodeler

# Install the dependencies for RepeatModeler

> conda update -c conda-forge perl-file-which

# Download the NINJA package for large-scale neighbor-joining phylogeny inference and clustering

> mkdir bin

> cd bin

> wget https://github.com/TravisWheelerLab/NINJA/archive/refs/tags/0.95-cluster_only.tar.gz

> tar -zxvf 0.95-cluster_only.tar.gz

> cd NINJA-0.95-cluster_only/NINJA/

> make # Create the ''Ninja'' executable file

> pwd

# The pwd Linux command prints the current working directory path

# The standard output of the "pwd" command will be used as a parameter of RepeatModeler
```

17. Repeat masking using known metazoan repeats with RepeatMasker:

```
#!/usr/bin/env bash

# You should use scaffold files as the input in RepeatMasker

for sample in `ls *fa`;do

RepeatMasker -species metazoa -s -parallel 10 -xsmall -alignments $sample

done

# -s: sensitive

# -parallel 10: number of threads

# -xsmall: softmasking, that is, change the repeat regions into lowercase, rather than N
```

```
# Output of RepeatMasker

your_genome_assembly.fa.masked # masked FASTA file

your_genome_assembly.fa.tbl # repeat summary
```

18. Identify previously unknown repeats in your genome assembly using RepeatModeler:

```
#!/usr/bin/env bash

#1. Create a Database for RepeatModeler

BuildDatabase -name CLR CLR_scaffold.fa

BuildDatabase -name ONT ONT_scaffold.fa

BuildDatabase -name Hifi Hifi_scaffold.fa

# -name: The name of the database to create

#2. Run RepeatModeler

RepeatModeler -database CLR -pa 10 -LTRStruct -ninja_dir /home/assembly/bin/NINJA-0.95-cluster_only/NINJA

RepeatModeler -database ONT -pa 10 -LTRStruct -ninja_dir /home/assembly/bin/NINJA-0.95-cluster_only/NINJA

RepeatModeler -database Hifi -pa 10 -LTRStruct -ninja_dir /home/assembly/bin/NINJA-0.95-cluster_only/NINJA

# -database: prefix name of the database that is used in the BuildDatabase function

# -pa: number of threads

# -LTRStruct: runs the LTR structural discovery pipeline for discovering LTR retrotransposons

# -ninja_dir: specify the NINJA folder
```

```
# Output of RepeatModeler

PREFIX-families.fa
```

19. Repeat masking with RepeatMasker using species-specific repeats that were found by RepeatModeler:

```
#!/usr/bin/env bash

RepeatMasker -lib CLR-families.fa -s -parallel 10 -xsmall -alignments CLR_scaffold.fa.masked

RepeatMasker -lib ONT-families.fa -s -parallel 10 -xsmall -alignments ONT_scaffold.fa.masked

RepeatMasker -lib Hifi-families.fa -s -parallel 10 -xsmall -alignments Hifi_scaffold.fa.masked

# -lib: specify your species-specific repeat FASTA file produced by RepeatModeler

# -s: sensitive

# -xsmall: softmasking, that is, change the repeat regions into lowercase, rather than N
```

```
# Output of RepeatMasker

your_genome_assembly.fa.masked.masked # masked FASTA file

your_genome_assembly.fa.masked.tbl # repeat summary
```

20. Conduct gene annotation.
    a. Map RNA sequencing reads to the masked genome:

```
#!/usr/bin/env bash

#1. Create the masked genome index

#Usage: hisat2-build repeat_masked_genome_assembly.fa PREFIX

hisat2-build CLR_scaffold.fa.masked.masked CLR

hisat2-build ONT_scaffold.fa.masked.masked ONT

hisat2-build Hifi_scaffold.fa.masked.masked Hifi

#2. Mapping RNA sequencing reads to the masked genome

hisat2 -x CLR -p 10 -1 /home/assembly/analysis/00_STARprotocol/SRR15130841_GSM5452671_
Control_CM1_Drosophila_melanogaster_RNA-Seq_1.fastq.gz -2 /home/assembly/analysis/00_
STARprotocol/SRR15130841_GSM5452671_Control_CM1_Drosophila_melanogaster_RNA-Seq_2.
fastq.gz | samtools sort -@ 10 -O BAM -o CLR.bam

hisat2 -x ONT -p 10 -1 /home/assembly/analysis/00_STARprotocol/SRR15130841_GSM5452671_
Control_CM1_Drosophila_melanogaster_RNA-Seq_1.fastq.gz -2 /home/assembly/analysis/00_
STARprotocol/SRR15130841_GSM5452671_Control_CM1_Drosophila_melanogaster_RNA-Seq_2.
fastq.gz | samtools sort -@ 10 -O BAM -o ONT.bam

hisat2 -x Hifi -p 10 -1 /home/assembly/analysis/00_STARprotocol/SRR15130841_GSM5452671_
Control_CM1_Drosophila_melanogaster_RNA-Seq_1.fastq.gz -2 /home/assembly/analysis/00_
STARprotocol/SRR15130841_GSM5452671_Control_CM1_Drosophila_melanogaster_RNA-Seq_2.
fastq.gz | samtools sort -@ 10 -O BAM -o Hifi.bam

# For HISAT2, the parameters are as follows:

# -x: index prefix

# -p: the number of threads HISAT2 will use

# -1 and -2: paired-end files. You can change the name of your sequencing data

# For SAMtools, the parameters are as follows:

# sort: SAMtools module to sort the mapped read information

# -@: the number of threads SAMtools will use

# -o: output file name

# -O BAM: output as a BAM format
```

    b. Make a conda environment for gene annotation:

```
# Type the following script directly in your terminal

# Create the conda environment for braker2

> conda create -n braker -c bioconda braker2

# Activate the environment for braker2

> conda activate braker

# Download GeneMark-EX program(gmes_linux_64.tar) and GeneMark key(gm_key_64) from http://
exon.gatech.edu/GeneMark/license_download.cgi (the GeneMark-ES/ET/EP) option

# Due to license and distribution restrictions, GeneMark and ProtHint should be separately
installed for BRAKER2 to become fully functional
```

```
#1. GeneMark-EX program

> tar -xvf gmes_linux_64.tar

> cd gmes_linux_64

> perl change_path_in_perl_scripts.pl "/usr/bin/env perl"

# This is required for BRAKER to accurately find the ".gm_key". See the "2. GeneMark key" section

> pwd

# The pwd Linux command prints the current working directory path

# Standard output of ''pwd'' command will be used parameter of braker

#2. GeneMark key

# GeneMark-EX will only run if a valid key file resides in your home directory

# The key file will expire after 200 days, which means that you have to download a new GeneMark-EX
release and a new key file after 200 days.

> cd # change to your home directory

> mv gm_key_64 .gm_key
```

c. Predict gene models using BRAKER:

```
#!/usr/bin/env bash

# Making working directory before the execution of braker program

mkdir CLR

braker.pl -genome=CLR_scaffold.fa.masked.masked -bam=CLR.bam -softmasking -cores 10
-workingdir=./CLR -GENEMARK_PATH=/home/assembly/bin/gmes_linux_64

mkdir ONT

braker.pl -genome=ONT_scaffold.fa.masked.masked -bam=ONT.bam -softmasking -cores 10
-workingdir=./ONT -GENEMARK_PATH=/home/assembly/bin/gmes_linux_64

mkdir Hifi

braker.pl -genome=Hifi_scaffold.fa.masked.masked -bam=Hifi.bam -softmasking -cores 10
-workingdir=./Hifi -GENEMARK_PATH=/home/assembly/bin/gmes_linux_64

# -cores 10: number of threads

# -bam: input BAM file which created by Hisat2

# -softmasking: repetitive sequences of the input genome is soft-masked

# -GENEMARK_PATH: specify the Genemark-EX program folder

# For more information about BRAKER: https://github.com/Gaius-Augustus/BRAKER
```

```
# Outputs of BRAKER

augustus.hints.aa # Amino acid FASTA sequences for your coding genes

augustus.hints.codingseq # Nucleotide FASTA sequences for your coding genes

augustus.hints.gtf # GTF file for your coding genes, which include their positions, orienta-
tion, and ID, etc.
```

Table 1. Output summary statistics for three different long-read sequencing platforms used in this paper

	PacBio CLR	PacBio HiFi	ONT
Number of reads	1,437,524	2,301,518	640,215
Read max (bp)	99,345	26,462	417,450
Read N50 (bp)	13,094	11,151	21,491
Read min (bp)	50	369	1
Total read length (bp)	12,016,661,679	25,600,110,705	7,133,020,037
Number of contigs	452	654	208
Contig max (bp)	23,607,911	24,502,687	27,938,801
Contig N50 (bp)	9,177,974	4,127,200	18,567,724
Contig min (bp)	1,381	9,867	21
Total contig length (bp)	141,740,149	168,692,738	133,002,022
Number of placed contigs	169	175	99
Length of placed contigs (bp)	134,417,363	142,931,941	130,673,188
Number of unplaced contigs	283	479	109
Length of unplaced contigs (bp)	7,322,786	25,760,797	2,328,834

EXPECTED OUTCOMES

Following this protocol, most draft genomes of multicellular organisms can be easily assembled. The contig N50 length and BUSCO metrics can be used to evaluate the quality of the *de novo* assembled genomes. In this guide, we outlined three assemblies using three different long-read sequencing platforms, namely, PacBio CLR, PacBio HiFi, and ONT (Figures 2B and 2C).

We conducted an analysis using publicly available *Drosophila melanogaster* data, which yielded a sufficiently large amount of data for assembly (Table 1). Cumulative contig length ratios were plotted in different line graphs, which showed that the N50 lengths differed (Figure 2B, solid black vertical line); in this case, ONT assembly was the best choice, followed by PacBio CLR and PacBio HiFi assemblies. BUSCO values were also compared in the three assemblies, indicating that assembly contiguity in terms of the number of complete single-copy orthologs was mostly contiguous in the PacBio CLR assembly, whereas the PacBio HiFi assembly exhibited some duplicated genes, and the ONT assembly exhibited more fragmented and missing genes than the other assemblies (Figure 2C).

Notably, the sequencing platform you choose for your genome assembly will depend on your sample and its genomic architecture, such as its genome size, heterozygosity, composition, and the number of repetitive elements. It is possible to visualize alignments between your assembly and the chromosome-level reference genome (Figure 3). The gray regions indicate that your scaffolds are well aligned to the reference, whereas the white regions indicate missing alignments and the yellow regions inverted alignments. Although HiFi assembly exhibited many gaps in the current case, its raw reads were generated from a heterozygous sample with an extremely low amount of input DNA (~10 ng).

SVs, which are difficult to be precisely detected using short-read sequencing technologies, can be detected more precisely using long-read sequencing technologies. In reality, by comparing our contigs to the reference genome, it is possible to detect SVs by category, e.g., by insertion, deletion, and inversion. When read- and assembly-based SV calling data are compared (Figures 4A and 4B, respectively), insertions are called more often in the assembly-based SV method because assembled genomes can cover much larger regions than each raw read. Finally, after the genome has been fully assembled, gene models can be annotated using the RNA-seq data. Our protocol is expected to aid research on intraspecies genome evolution as it will facilitate genome alignments and the detection of SVs.

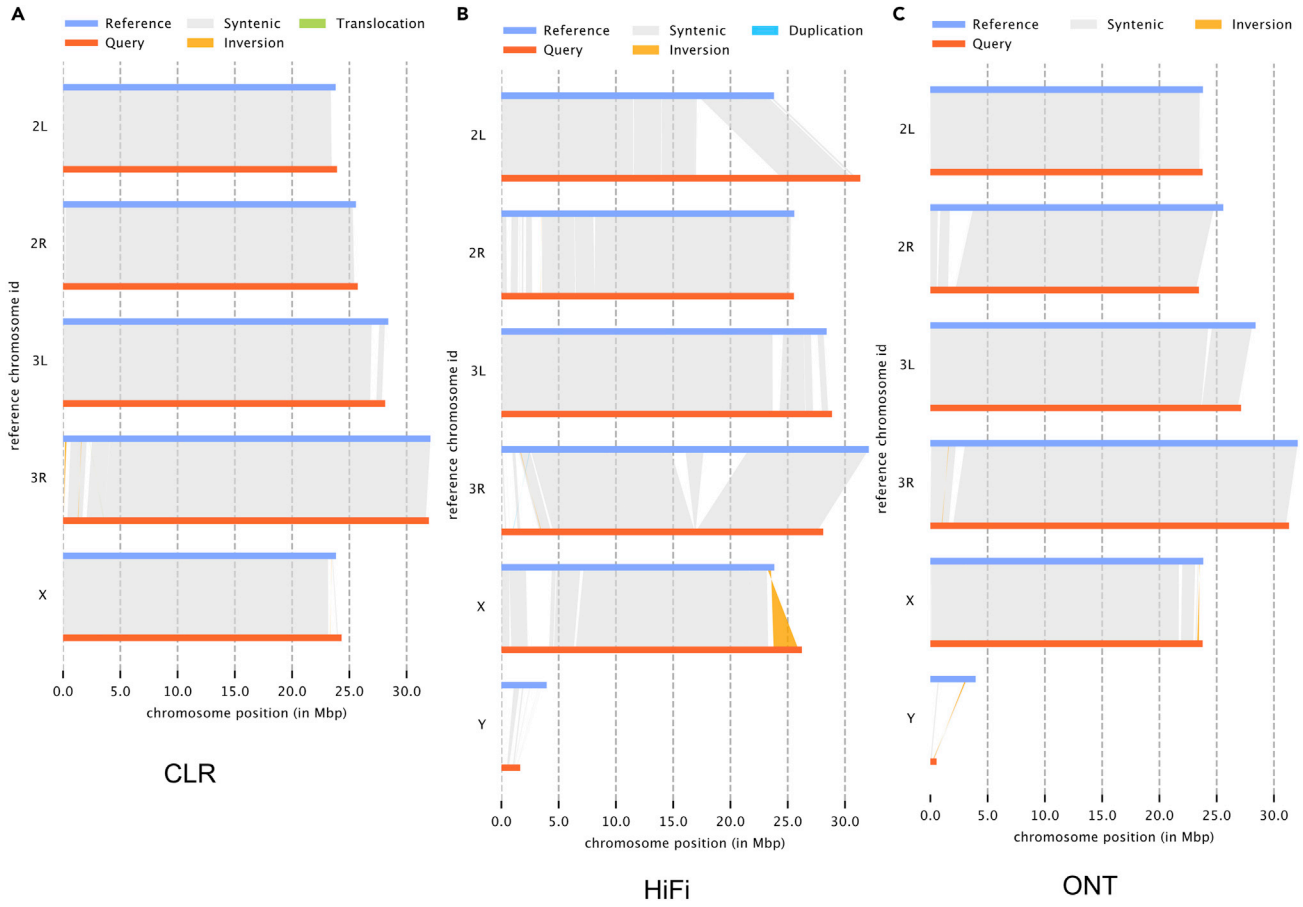


Figure 3. Visualization of synteny between your genome assembly (Query) and a reference genome (Reference)

(A) Scaffold derived from PacBio CLR data.

(B) Scaffold derived from PacBio HiFi data.

(C) Scaffold derived from ONT data.

LIMITATIONS

In this guide, we did not cover SNP calling, isoform detection, and scaffolding without a chromosome-level reference genome. The procedure for calling SNPs has been thoroughly described elsewhere (Bellinger, 2020; Koboldt, 2020). Scaffolding is the process by which contigs are joined together to construct pseudo-chromosome-level genome assemblies. To complete scaffolding, you will need additional datasets, such as physical (e.g., Hi-C), optical (Bionano), and genetic mapping data. Notably, you may lose many isoforms when using our protocol as short-read RNA sequencing data are too short to identify full-length isoforms. Furthermore, our protocol may not completely address isoform information. It would be preferable to annotate isoform information using full-length transcript sequencing data based on long-read RNA sequencing data rather than short-read RNA sequencing data. Finally, assembling the polyploidy genome with the current technology is challenging; thus, our protocol does not cover the polyploidy genome.

TROUBLESHOOTING

Problem 1

The code block cannot be executed even though the program required for the protocol is installed.

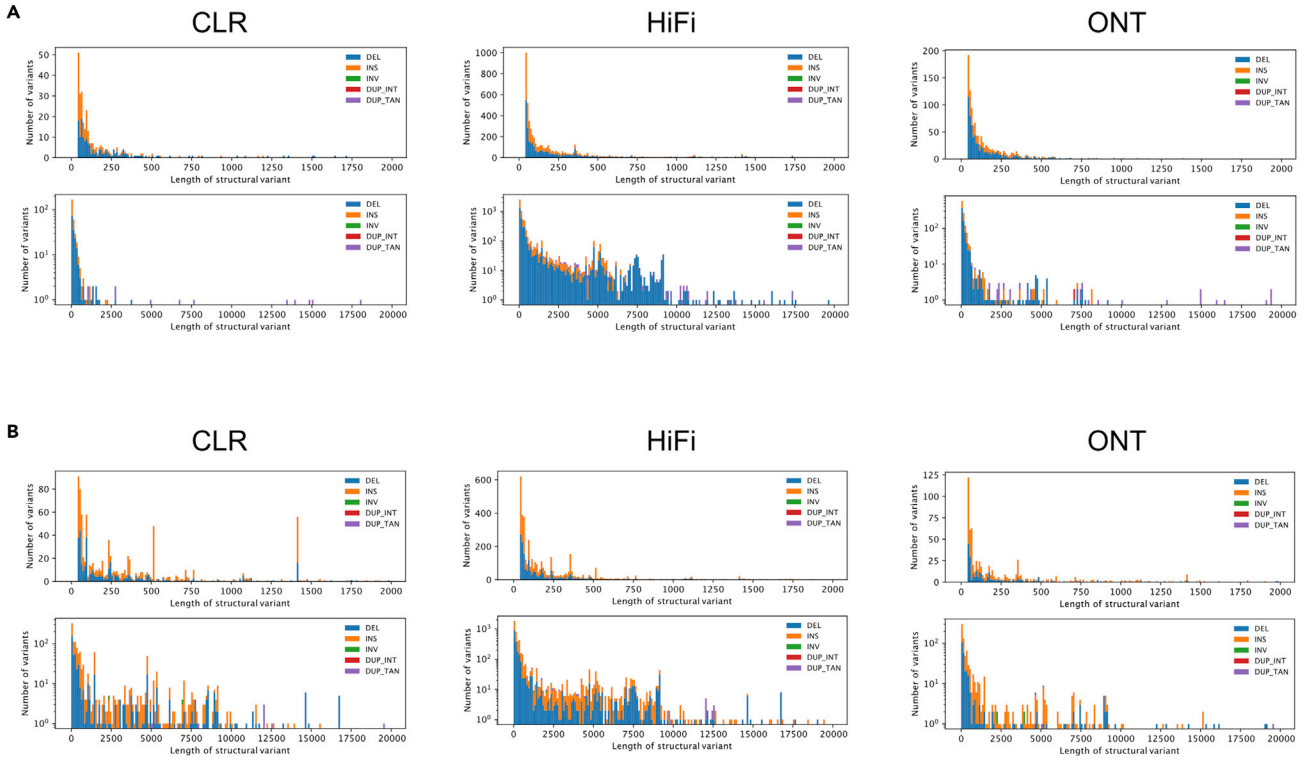


Figure 4. SV calling output summary

(A) Read-based SV calling with SVIM.

(B) Assembly-based SV calling with SVIM-asm.

Potential solution

First, look carefully at the error messages in the console. In many cases, the problem arises because the conda environment has not been activated or the location of the input required for execution has not been correctly specified. We recommend that you create a separate analysis folder for each analysis and bring the input file required for analysis as a symbolic link (*ln -s* command in Linux). When running R scripts, you must designate the working directory (*setwd* command in R) or run R script in the folder where the input file is located. Second, check your code for typos. There are often typos in quotes, commas, and input names.

Problem 2

The installation of conda is taking a long time.

Potential solution

Conda can run into endless loops when it cannot solve the dependencies with the packages that have been previously installed. If you encounter this problem, we recommend creating another separate conda environment or considering mamba as a replacement for conda (https://github.com/mamba-org/mamba). Mamba is a reimplementation of the conda package manager in C++, and the commands of mamba are nearly identical to those of conda, except conda should be replaced with mamba in the commands. When installing programs with several dependencies, such as RepeatModeler or BRAKER, we strongly recommend using mamba. If it does not solve the problem, it is possible that many packages are already installed in your base environment. We recommend either deleting the package in the base environment or asking the server administrator for a new ID.

```
# Example of mamba usage

# Install mamba into the base environment

> conda install mamba -n base -c conda-forge

# Create the conda environment for braker2 using mamba command

> mamba create -n braker -c bioconda braker2
```

Problem 3

My species is too small to obtain a sufficient amount of DNA.

Potential solution

Currently, ONT and PacBio require >1 and >3 μg of DNA, respectively. Such amounts may not be fulfilled when using very small animals, including nematodes. For nematodes, we typically culture the animals into inbred or sibling-bred lines; however, many other animals cannot be cultured. For small species, you can consider the low (400 ng) or ultra-low (5 ng) DNA input sequencing protocols available in PacBio HiFi sequencing if your species has a genome size of <1 Gb or <500 Mb, respectively (https://www.pacb.com/wp-content/uploads/Application-Note-Considerations-for-Using-the-Low-and-Ultra-Low-DNA-Input-Workflows-for-Whole-Genome-Sequencing.pdf). For example, Kingan et al., 2019 demonstrated that a high-quality genome can be assembled using a single mosquito (Kingan et al., 2019). However, standard high-input DNA sequencing would likely be a better choice than these low-input protocols if sufficient DNA is available.

Problem 4

The contig N50 length is too short.

Potential solution

It would be preferable to check your species' ploidy level, estimate the genome size with independent experiments, and generate additional high-quality long-read sequencing data. Using long-read sequencing technologies, it remains difficult to resolve extremely long segmental duplication blocks and highly clustered repetitive sequences that can span hundreds of kilobases. Given that even diploid genomes can become problematic, haploid or inbreeding lines would be the best choice for de novo genome assembly, and polyploidy genomes should be avoided. The first gap-free complete human genome, for example, was assembled using a human haploid cell line derived from a complete hydatidiform mole (Nurk et al., 2022). To resolve interspersed repeats or segmental duplication blocks in PacBio CLR and ONT data, the read N50 length should be >10 kb. Furthermore, the genome size can be estimated using sequencing data independently via flow cytometry or real-time PCR (Hare and Johnston, 2012; Wilhelm et al., 2003). Your genome assembly could be too fragmented for unknown reasons; additional sequencing may be beneficial but not always.

Problem 5

Synteny analysis does not work.

Potential solution

To run SyRi, a synteny analysis tool, the two genomes must have the same number of chromosomes and the same name. Aside from the chromosomal name, the reference genome downloaded from a specific database may contain additional information attached to the FASTA header. In this case, the $name variable built into the bioawk application can be used to simply reformat the FASTA file.

RESOURCE AVAILABILITY

Lead contact

Further information and requests for resources and reagents should be directed to and will be fulfilled by the lead contact, Chuna Kim (kimchuna@kribb.re.kr).

Materials availability

This study did not generate new unique reagents.

Data and code availability

This protocol did not generate any new datasets. The key resources table contains all the accession numbers for the sample data analyzed in this protocol. All codes used for data analysis are included in this manuscript.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea (NRF-2020R1C1C101220611, NRF-2019R1A6A1A10073437) and National Research Council of Science & Technology (NST) Aging Convergence Research Center (CRC22011-300).

AUTHOR CONTRIBUTIONS

Conceptualization, J.K. and C.K.; methodology and formal analysis, J.K. and C.K.; data curation, J.K.; writing, J.K. and C.K.; all authors have read and agreed to the published version of the manuscript.

DECLARATION OF INTERESTS

The authors declare no competing interests.

REFERENCES

Alonge, M., Lebeigle, L., Kirsche, M., Aganezov, S., Wang, X., Lippman, Z., Schatz, M., and Soyk, S. (2021). Automated assembly scaffolding elevates a new tomato system for high-throughput genome editing. Preprint at BioRxiv. https://doi.org/10.1101/2021.11.18.469135.

Bellinger, M.R. (2020). SNP Calling and VCF Filtering Pipeline. protocols.io. https://doi.org/10.17504/protocols.io.84fhytn.

Cheng, H., Concepcion, G.T., Feng, X., Zhang, H., and Li, H. (2021). Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. Nat. Methods *18*, 170–175. https://doi.org/10.1038/s41592-020-01056-5.

The Darwin Tree of Life Project Consortium (2022). Sequence locally, think globally: the Darwin tree of life Project. Proc. Natl. Acad. Sci. U S A *119*. e2115642118. https://doi.org/10.1073/pnas.2115642118.

Goel, M., Sun, H., Jiao, W.-B., and Schneeberger, K. (2019). SyRI: finding genomic rearrangements and local sequence differences from whole-genome assemblies. Genome Biol. *20*, 277. https://doi.org/10.1186/s13059-019-1911-0.

Grabherr, M.G., Haas, B.J., Yassour, M., Levin, J.Z., Thompson, D.A., Amit, I., Adiconis, X., Fan, L., Raychowdhury, R., Zeng, Q., et al. (2011). Trinity: reconstructing a full-length transcriptome without a genome from RNA-Seq data. Nat. Biotechnol. *29*, 644–652. https://doi.org/10.1038/nbt.1883.

Hare, E.E., and Johnston, J.S. (2012). Genome size determination using flow cytometry of propidium iodide-stained nuclei. In Molecular methods for evolutionary genetics (Springer), pp. 3–12.

Heller, D., and Vingron, M. (2019). SVIM: structural variant identification using mapped long reads. Bioinformatics *35*, 2907–2915. https://doi.org/10.1093/bioinformatics/btz041.

Heller, D., and Vingron, M. (2020). SVIM-asm: structural variant detection from haploid and diploid genome assemblies. Bioinformatics *36*, 5519–5521. https://doi.org/10.1093/bioinformatics/btaa1034.

Hoff, K.J., Lange, S., Lomsadze, A., Borodovsky, M., and Stanke, M. (2016). BRAKER1: unsupervised RNA-seq-based genome annotation with GeneMark-ET and AUGUSTUS: table 1. Bioinformatics *32*, 767–769. https://doi.org/10.1093/bioinformatics/btv661.

Jain, M., Koren, S., Miga, K.H., Quick, J., Rand, A.C., Sasani, T.A., Tyson, J.R., Beggs, A.D., Dilthey, A.T., Fiddes, I.T., et al. (2018). Nanopore sequencing and assembly of a human genome with ultra-long reads. Nat. Biotechnol. *36*, 338–345. https://doi.org/10.1038/nbt.4060.

Kim, C., Kim, J., Kim, S., Cook, D.E., Evans, K.S., Andersen, E.C., and Lee, J. (2019). Long-read sequencing reveals intra-species tolerance of substantial structural variations and new subtelomere formation in C. elegans. Genome research *29*, 1023–1035.

Kim, E., Kim, J., Kim, C., and Lee, J. (2021). Long-read sequencing and de novo genome assemblies reveal complex chromosome end structures caused by telomere dysfunction at the single nucleotide level. Nucleic acids research *49*, 3338–3353.

Kim, D., Paggi, J.M., Park, C., Bennett, C., and Salzberg, S.L. (2019). Graph-based genome alignment and genotyping with HISAT2 and HISAT-genotype. Nat. Biotechnol. *37*, 907–915. https://doi.org/10.1038/s41587-019-0201-4.

Kim, C., Sung, S., Kim, J., and Lee, J. (2020). Repair and reconstruction of telomeric and subtelomeric regions and genesis of new telomeres: implications for chromosome evolution. Bioessays *42*, 1900177.

Kingan, S.B., Heaton, H., Cudini, J., Lambert, C.C., Baybayan, P., Galvin, B.D., Durbin, R., Korlach, J., and Lawniczak, M.K. (2019). A high-quality de novo genome assembly from a single mosquito using PacBio sequencing. Genes *10*, 62.

Koboldt, D.C. (2020). Best practices for variant calling in clinical sequencing. Genome Medicine *12*, 1–13.

Koren, S., Walenz, B.P., Berlin, K., Miller, J.R., Bergman, N.H., and Phillippy, A.M. (2017). Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. Genome Res. *27*, 722–736. https://doi.org/10.1101/gr.215087.116.

Lewin, H.A., Robinson, G.E., Kress, W.J., Baker, W.J., Coddington, J., Crandall, K.A., Durbin, R., Edwards, S.V., Forest, F., Gilbert, M.T.P., et al. (2018). Earth BioGenome Project: sequencing life for the future of life. Proc. Natl. Acad. Sci. U S A *115*,

4325–4333. https://doi.org/10.1073/pnas.1720115115.

Li, H. (2017). BWK Awk Modified for Biological Data (GitHub). https://github.com/lh3/bioawk.

Li, H. (2021). New strategies to improve minimap2 alignment accuracy. Bioinformatics *37*, 4572–4574. https://doi.org/10.1093/bioinformatics/btab705.

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., and Durbin, R. (2009). The sequence alignment/map format and SAMtools. Bioinformatics *25*, 2078–2079. https://doi.org/10.1093/bioinformatics/btp352.

Manni, M., Berkeley, M.R., Seppey, M., Simão, F.A., and Zdobnov, E.M. (2021). BUSCO update: novel and streamlined workflows along with broader and deeper phylogenetic coverage for scoring of eukaryotic, prokaryotic, and viral genomes. Mol. Biol. Evol. *38*, 4647–4654. https://doi.org/10.1093/molbev/msab199.

Mapleson, D., Garcia Accinelli, G., Kettleborough, G., Wright, J., and Clavijo, B.J. (2017). KAT: a K-mer analysis toolkit to quality control NGS datasets and genome assemblies. Bioinformatics *33*, 574–576. https://doi.org/10.1093/bioinformatics/btw663.

Nurk, S., Koren, S., Rhie, A., Rautiainen, M., Bzikadze, A.V., Mikheenko, A., Vollger, M.R., Altemose, N., Uralsky, L., Gershman, A., et al. (2022). The complete sequence of a human genome. Science *376*, 44–53. https://doi.org/10.1126/science.abj6987.

R Core Team. (2013). R: A language and environment for statistical computing.

Shafin, K., Pesout, T., Lorig-Roach, R., Haukness, M., Olsen, H.E., Bosworth, C., Armstrong, J., Tigyi, K., Maurer, N., Koren, S., et al. (2020). Nanopore sequencing and the Shasta toolkit enable efficient de novo assembly of eleven human genomes. Nat. Biotechnol. *38*, 1044–1053. https://doi.org/10.1038/s41587-020-0503-6.

Smit, A., Hubley, R., and Green, P. (2013–2015). RepeatMasker open-4.0. http://www.repeatmasker.org.

Smit, A., and Hubley, R. (2008–2015). RepeatModeler open-1.0. http://www.repeatmasker.org.

Wellcome Sanger Institute Pathogen Informatics. (2020) (Wellcome Sanger Institute Pathogen Informatics). https://github.com/sanger-pathogens/assembly-stats.

Wenger, A.M., Peluso, P., Rowell, W.J., Chang, P.-C., Hall, R.J., Concepcion, G.T., Ebler, J., Fungtammasan, A., Kolesnikov, A., Olson, N.D., et al. (2019). Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. Nat. Biotechnol. *37*, 1155–1162. https://doi.org/10.1038/s41587-019-0217-9.

Wickham, H. (2007). Reshaping data with the reshape package. J. Stat. Softw. *21*, 1–20. https://doi.org/10.18637/jss.v021.i12.

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L.D.A., François, R., François, R., Grolemund, G., Hayes, A., Henry, L., et al. (2019). Welcome to the tidyverse. J. Open Source Softw. *4*, 1686. https://doi.org/10.21105/joss.01686.

Wickham, H., Chang, W., and Wickham, M.H. (2016). Package 'ggplot2'. Create Elegant Data Visualisations Using the Grammar of Graphics Version *2*, pp. 1–189.

Wickham, H., François, R., and Henry, L. (2021). Müller K. Dplyr: A Grammar of Data Manipulation. R package version 08 4.

Wilhelm, J., Pingoud, A., and Hahn, M. (2003). Real-time PCR-based method for the estimation of genome sizes. Nucleic Acids Research *31*, e56.

Wilke, C.O. (2019). Cowplot: Streamlined Plot Theme and Plot Annotations for "Ggplot2. R package version 1.