Alternative-splicing detection by NGS

walkthroughs

Wen-Dar Lin

Bioinformatics core, IPMB

wdlin@gate.sinica.edu.tw

Preface

- This presentation file is to describe key steps and important notes
  - stringtie pipeline
  - rackj pipeline

Disclaimer

- This part is intended to give useful logs to users who has experiences on using linux environment for computation.
- For those who are not familiar with linux operation, you may take a look at explanations of key steps.
- CAUTION: Better not copy command from this PowerPoint file. Office might twist symbols like - ' ".

Files & and how to use them

- Datafile:
  - 1. AS depositar: https://data.depositar.io/en/dataset/exampledata-zip-20260922
  - OR 2. Figshare: https://doi.org/10.6084/m9.figshare.33773359
  - about 750MB
  - Please download and unzip it
  - There should be an “ExampleData” folder. **Remember where this folder is**.
  - If you are using linux system, execute “chmod -R 755 ExampleData” to make sure the folder writable

Files & and how to use them

- Docker image:
  - We provide a docker image
    - so that you can install just *docker desktop*, *docker*, or *singularity* and use the image to create an exactly-the-same environment.
  - Next two pages for
    - links to install docker program, and
    - the commands to create the environment

Files & and how to use them

- Docker desktop
  - For computer with a graphical user interface (ex: win11 and macOS)
    - Windows: https://docs.docker.com/desktop/setup/install/windows-install/
    - Mac: https://docs.docker.com/desktop/setup/install/mac-install/
    - Linux: https://docs.docker.com/desktop/setup/install/linux/
- Docker
  - For linux command-line interface
  - https://docs.docker.com/engine/install/
- SingularityCE 3.x (or Apptainer 1.x)
  - Usually available in HPC environment
  - Ask your system admin for it

Files & and how to use them

- Commands to create the environment
  - Docker desktop
    - Open *terminal* (bottom-right corner of Docker Desktop) and enter:
      docker run -it --rm --mount type=bind,src=**<path_to_ExampleData>**,dst=/mnt wdlin/rackj
  - Docker (command line)
    - sudo docker run -it --rm --mount type=bind,src=**<path_to_ExampleData>**,dst=/mnt wdlin/rackj
  - Singularity
    - singularity run --bind "**<path_to_ExampleData>**:/mnt" docker://wdlin/rackj

The example dataset

- Randomly (1%) extracted from publicly available SRA dataset (SRP071829)
  - created by Dr. Matzke's lab, IPMB
  - triplicates of control and treatment Arabidopsis samples
    - a total of 6 samples
  - strand specific pair-ended RNAseq
    - a total of 12 FASTQ files

Walkthroughs

- We assume that the computing environment was reproduced by using the docker image
  - No steps for setup in this slides
- Walkthroughs and detailed explanations can be found at https://github.com/wdlingit/rackj/tree/main/walkthrough

StringTie pipeline

- Running tophat2, build indexes
  - In addition of building genome index, it is strongly suggest to build transcriptome index by Tophat2 manual.
    - Avoiding race condition and saving time.

```
### running tophat2

# bowtie2 build genome index, this takes time
Singularity> bowtie2-build TAIR10_chr_all.fas tair10.genome

# tophat2 build transcriptome index, this takes time
Singularity> tophat2 -G TAIR10_GFF3_genes_transposons.gff --
transcriptome-index=tair10.transcriptome/known tair10.genome
```

StringTie pipeline

- Running tophat2, mapping reads
  - The same “ls src/*.fq.gz | sort” + perl oneliner for executing tophat for the 6 samples
  - NOTE: you may switch to any other mapping tools as you like. MUST refer StringTie official website!

```
# align reads using tophat2, guided with tair10 annotation

Singularity> ls src/*.fq.gz | sort | perl -ne 'chomp;
/.+\/(.+)_R\d\./; push @{$hash{$1}},$_; if(eof){ for $k
(sort keys %hash){ $cmd="tophat2 -o $k"."_tophat2 -p 4 --
transcriptome-index=tair10.transcriptome/known tair10.genome
@{$hash{$k}}"; print "\nCMD: $cmd\n"; system $cmd } }'
```

StringTie pipeline

- Running stringtie programs
  - It seems that stringtie doesn't recognize TAIR10 GFF3 file well. So I decide to translate this GFF3 file into a GTF file.

```
# translate TAIR10 GFF3 into GTF

Singularity> gffread -T -o tair10.gtf
TAIR10_GFF3_genes_transposons.gff 2>/dev/null
```

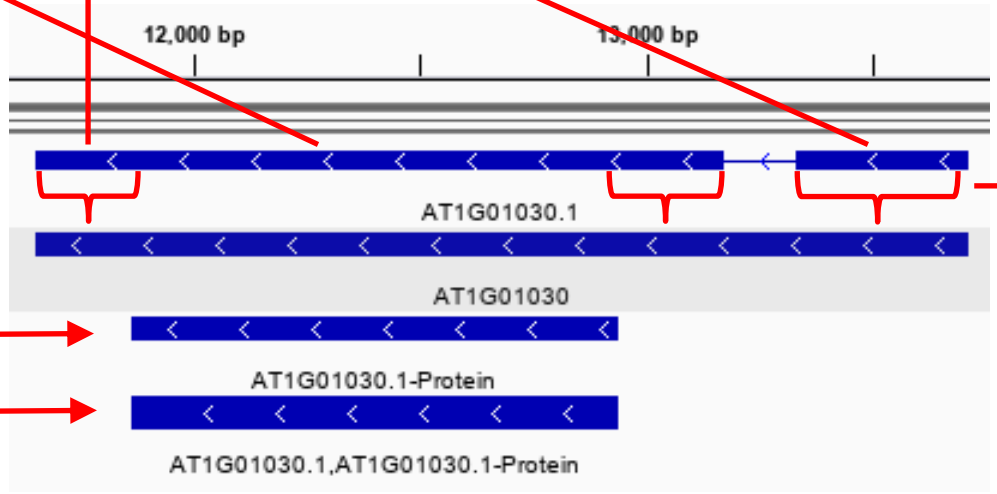
The GFF3 format

- It is common to see that genome annotations are stored in a GFF3 format file
  - Usually in download area of the genome's official website, which should be with the genome's FASTA file.
    - NOTE: if there is a README file, must check it.
- A GFF3 file storing genome annotation tells you *which genes are at where of the genome*.

The GFF3 format

- A GFF3 file also stores hierarchy of recorded objects.

Chr1	gene	11649	13714	ID=AT1G01030
Chr1	mRNA	11649	13714	ID=AT1G01030.1;Parent=AT1G01030
Chr1	protein	11864	12940	ID=AT1G01030.1-Protein;Derives_from=AT1G01030.1
Chr1	five_prime_UTR	13335	13714	Parent=AT1G01030.1
Chr1	exon	13335	13714	Parent=AT1G01030.1
Chr1	five_prime_UTR	12941	13173	Parent=AT1G01030.1
Chr1	CDS	11864	12940	Parent=AT1G01030.1-Protein;
Chr1	three_prime_UTR	11649	11863	Parent=AT1G01030.1
Chr1	exon	11649	13173	Parent=AT1G01030.1



The GFF3 format

- From *parent-child relationships* in the last slide, we can conclude the following tree structure of annotation features.
- So a GFF3 file is telling
  - *which genes are at where of the genome,*
  - *which transcripts are derived from which genes, and*
  - *exon coordinates of transcripts.*

```
gene*
    mRNA*
       protein*
          CDS
       exon
       five_prime_UTR
       CDS
       three_prime_UTR
```

The GTF format

- Can be considered as a previous version of GFF3
  - which usually stores only exon regions
  - plus gene id and transcript id for each exon.

StringTie pipeline

- Running stringtie programs
  - In this very first step, what we have to do is to use stringtie to build one assembly for each sample with the aid of know genome annotation

sample A

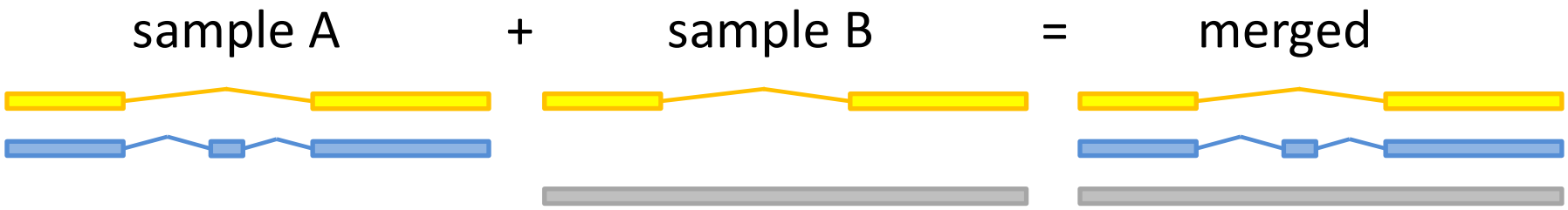
sample B

```
# stringtie, guided assembly

Singularity> ls *_tophat2/accepted_hits.bam | perl -ne
'chomp; /(.+?)_tophat2/; $cmd="stringtie $_ -o $1.gtf -p 4 -
G tair10.gtf"; print "\nCMD: $cmd\n"; system $cmd'
```

StringTie pipeline

- Running stringtie programs
  - Comparison between samples means that we need a unified transcriptome. So use “stringtie --merge” to combine assemblies.




```
# stringtie, merge mode, combine assemblies of replicates
into one master transcriptome

Singularity> stringtie --merge -G tair10.gtf -o merged.gtf
control_rep1.gtf control_rep2.gtf control_rep4.gtf
treatment_rep5.gtf treatment_rep7.gtf treatment_rep9.gtf
```

StringTie pipeline

- Running stringtie programs
  - Above steps are for assembly generation. With merged assembly, use stringtie with option “-eB” to produce counts for transcripts.
  - NOTE: -o must be assigned to a separate subfolder for each sample because stringtie are outputting all count files with exactly the same names.


```
# stringtie, generate tables

Singularity> ls *_tophat2/accepted_hits.bam | perl -ne
'chomp; /(.+?)_tophat2/; $cmd="stringtie $_ -eB -o $1/$1.gtf
-p 4 -G merged.gtf"; print "\nCMD: $cmd\n"; system $cmd'
```

StringTie pipeline

- Count files in the same names

```
Singularity> ls -l */*.ctab
-rw-rw-r-- 1 ubuntu ubuntu  2826155 Oct  5 17:11 control_rep1/e2t.ctab
-rw-rw-r-- 1 ubuntu ubuntu 11682721 Oct  5 17:11 control_rep1/e_data.ctab
-rw-rw-r-- 1 ubuntu ubuntu  2246980 Oct  5 17:11 control_rep1/i2t.ctab
-rw-rw-r-- 1 ubuntu ubuntu  5092894 Oct  5 17:11 control_rep1/i_data.ctab
-rw-rw-r-- 1 ubuntu ubuntu  3554980 Oct  5 17:11 control_rep1/t_data.ctab
-rw-rw-r-- 1 ubuntu ubuntu  2826155 Oct  5 17:11 control_rep2/e2t.ctab
-rw-rw-r-- 1 ubuntu ubuntu 11684916 Oct  5 17:11 control_rep2/e_data.ctab
-rw-rw-r-- 1 ubuntu ubuntu  2246980 Oct  5 17:11 control_rep2/i2t.ctab
-rw-rw-r-- 1 ubuntu ubuntu  5093349 Oct  5 17:11 control_rep2/i_data.ctab
-rw-rw-r-- 1 ubuntu ubuntu  3555067 Oct  5 17:11 control_rep2/t_data.ctab
-rw-rw-r-- 1 ubuntu ubuntu  2826155 Oct  5 17:11 control_rep4/e2t.ctab
-rw-rw-r-- 1 ubuntu ubuntu 11679732 Oct  5 17:11 control_rep4/e_data.ctab
-rw-rw-r-- 1 ubuntu ubuntu  2246980 Oct  5 17:11 control_rep4/i2t.ctab
-rw-rw-r-- 1 ubuntu ubuntu  5092408 Oct  5 17:11 control_rep4/i_data.ctab
-rw-rw-r-- 1 ubuntu ubuntu  3554974 Oct  5 17:11 control_rep4/t_data.ctab
-rw-rw-r-- 1 ubuntu ubuntu  2826155 Oct  5 17:11 treatment_rep5/e2t.ctab
-rw-rw-r-- 1 ubuntu ubuntu 11682131 Oct  5 17:11 treatment_rep5/e_data.ctab
-rw-rw-r-- 1 ubuntu ubuntu  2246980 Oct  5 17:11 treatment_rep5/i2t.ctab
-rw-rw-r-- 1 ubuntu ubuntu  5093373 Oct  5 17:11 treatment_rep5/i_data.ctab
-rw-rw-r-- 1 ubuntu ubuntu  3554953 Oct  5 17:11 treatment_rep5/t_data.ctab
```

StringTie pipeline

- Running stringtie programs
  - The logic of stringtie is to collect counts in all those count files to some other program for differential analysis.
  - We use “prepDE.py” (comes with stringtie package) to generate a table of isoform read counts and sent it to DESeq2 for differential analysis.

```
# generate gene read counts/transcript read counts

Singularity> prepDE.py

Singularity> ls *.csv
gene_count_matrix.csv  transcript_count_matrix.csv
```

StringTie pipeline

- Check https://github.com/wdlingit/rackj/blob/main/walkthrough/Stringtie_short_reads.md#4-isoform-expression-level-comparison for rest parts.

rackj pipeline

- The rackj walkthrough at GitHub provides detailed explanations for steps and output files.
  - https://github.com/wdlingit/rackj/blob/main/walkthrough/Rackj_short_reads_AScomp.md
  - Check “Outline” for sections inside the walkthrough.