RNA-seq Analysis Setup and Prerequisites

Alec Taylor
aztaylor@ucsb.edu

Contents

1 Outline of RNA-seq Analysis Pipeline

1. Retrieving reference genomes and transcriptomes.
   - There are several ways to retrieve reference material, the one we are going to focus on is the SRA-toolkit.
   - The Sequence Read Archive (SRA) is a National Center for Biotechnology Information (NCBI) database
     which is part of the International Nucleotide Sequence Database Collaboration (INSDC).
   - It grants access to experimental sequencing data which is considered suitable for public distribution through run accessions numbers (typically in the for SRR*).
   - Run accession numbers are obtained by searching the SRA website.
2. Read trimming.
   - Trimming is necessary to clip sequencing addapters and bad reads (you may notice Ns in place of bad reads from illumina FastQ files).
   - This operation is used on the fastaQ files obtained from next generation sequencing (NGS) runs.
   - We will use Trimmomatic [BLU14], an illumina focused trimming tool.
3. Read QC on Raw FastQ files.
   - Read Quality Control typically assesses the quality of read mapping by creating mapping and coverage statistics.

- Aditionally Variant Calling and Read Calibration could current for errors from sequencing or transcription. Since we are not interested in classifying transcripts, we can can recalibrate these errors so they are not penalized and under represented in our Analysis/

4. Read quantification.
   - Quantification can be done using several tools. Examples of popular quantification software are provided below.
     (a) Sailfish[PMK14]
     (b) Salmon[Pat+17]
     (c) Kallisto[Bra+16]
     (d) RSEM[LD11]
   - All of the tools listed above are fairly similar in terms of accuracy and performance, with differences mainly occurring with fringe scenarios [Zha+17].
   - Sailfish, Salmon, and Kallisto are alignment independent tools, while RSEMs, when used in transcript quantification is alignment-dependent.
   - The primary factors to consider when choosing between the highlighted tools is therefore the presence of the fringe scenarios (this topic is explored in more detail in the following section) and runtime and computational capability.
   - Generally, alignment independent methods are faster and require less computational resources than alignment-dependent methods, while maintaining similar performance profiles [Zha+17].
   - It's worth noting that the developers tend to insist that their software is the top performer in their respective papers. If we look at comparison studies using simulated data where the ground truth is known, each shows it's own unique and often insignificant differences.
   - Generally, researches tend to choose either Salmon or Kallisto due to their speed advantage and accuracy, or sailfish for it's simplicity.
   - With this pipeline we will consider both Salmon and Kallisto.
5. Remove Duplicates
6. Base Recalibration

2 Comparison of Transcript Quantification Tools

Generally, transcript quantification tools can be divided into too categories, alignment-dependent or alignment
-independent tools. Considering was main driving motivation for the development of align-independent methods, another significant motivation was the advancement of transcript-level quantification. The need for transcript-level quantification came with the realization that RNA could encode for two or more exons (called isoforms) through eukaryotic alternative splicing. Therefore, annotated genes could be represented in various exons (gene isoforms). With gene-level quantification reads are mapped to the genome, and isoforms will either be quantified based on gene-mapping or discarded. Align-independent allows for quantification of individual isoforms. Since transcript-level quantification attemps to quantify isoforms and determine which genes are responsible for their formation , giving more insight into true gene experession. Unfortunately, this problem tends to be difficult, so much so that the algorithms rely on statistical algorithms to determine the probability of a read matching an gene i

Sailfish, Salmon, and Kallisto are examples of alignment-independent tools. They utilize psuedoalignment to improve speed by exploiting the idea that precise alignments aren't necessary to accuratly quantify reads. This is in contrast alignment-dependent tools, such as RSEM, which rely on pre-alliged outputs from external tools (e.g. STAR). The additional time and computational requirments pre-alignment has resulted in alignment-dependent tools falling out of favor. For this reason we willSalmon has two run modes, an alignment-dependent mode and an alignment independent mode. In terms of performance, all of the above quantification tools exhibit similarly high accuracy and robustness [Zha+17]. The one caveat is when dealing with low-abundance or short reads, with Between the alignment-free

Info: While Salmon, Sailfish, and Kalisto are primarily viewed as alignment-independent quantification tool, they can conduct alignment-dependent Quantification.

3 Useful Commands

3.1 bash/zsh

1. cd (change directory) - Changes the current working directory in terminal. Can take relative or full paths. For example if you are currently in the home directory ( ) and would like to move to the documents directory, you could use either:

   Zsh/Bash
   ```
   $ cd Documents
   or
   $ cd /Users/<your username>/Documents

   # where "/" represents the root directory containing all other directories.
   ```

2. ls

   Zsh/Bash
   ```
   $ cd Documents
   or
   $ cd /Users/<your username>/Documents

   where "/" represents the root directory containing all other directories.
   ```

3. sudo

   Zsh/Bash
   ```
   $ cd Documents
   or
   $ cd /Users/<your username>/Documents

   where "\/" represents the root directory containing all other directories.
   ```

4. curl

   Zsh/Bash
   ```
   $ cd Documents
   or
   $ cd /Users/<your username>/Documents

   where "/" represents the root directory containing all other directories.
   ```

5. mv

   Zsh/Bash
   ```
   $ cd Documents
   or
   $ cd /Users/<your username>/Documents

   where "/" represents the root directory containing all other directories.
   ```

6. rm

   Zsh/Bash
   ```
   $ cd Documents
   or
   $ cd /Users/<your username>/Documents

   where "/" represents the root directory containing all other directories.
   ```

7. sudo

3.2 git

4 Software Installation

4.1 Building From Scratch

4.1.1 Prerequisite Software

1. Xcode Release Candidate 15 and Homebrew.

   Info: Xcode Command Line Tools can also be installed in the macOS App Store, it comes with the Xcode app. To most, it's perefereable to only have the Xcode Developer Command Line Tools, because the Xcode app/IDE is generally recieved poorly.

   Another alternative is to download the most recent version of The most recent version of Xcode Developer. You will need an icloud account to sign in.

   Zsh/Bash

   ```
   # The argument -p or --print-path can be used to see any instances of xcode installed on your machine.
   $ xcode-select -p
   /Applications/Xcode.app/Contents/Developer # If the app is installed.
   # or
   $ xcode-select -p
   /Library/Developer/CommandLineTools # If you only have the command line tools installed.
   # and
   $ xcode-select -p
   xcode-select: error: unable to get active developer directory, use ‘sudo xcode-select --switch path/to/Xcode.app‘ to set
         one (or see ‘man xcode-select‘ # If neither the the app or command line tools are installed.

   # If you have the app installed, you should check if the command line tools are installed:
   $ ls /Library/Developer/
   CommandLineTools # If installed.
   ls: /Library/Developer: No such file or directory # If not installed.

   # In the case where you have only the app installed or neither, you should install the developer command line tools.
   # This can be accomplished in two ways:
   # Way 1
   $ xcode-select --install
   $ xcode-select -p
   $ /bin/bash -c "$(curl␣-fsSL␣https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
   Password:

   # Which is actually installing Homebrew (a package manager), but installs Xcode Developer Command Line Tools as a
         dependency.

   # Way 2
   $ xcode-select --install
   xcode-select: note: install requested for command line developer tools.

   # The first option to download the command tools is nice because Homebrew is a popular and useful package manager for
         macOS. The disadvantage is that this is a large package, especially when bundled with Xcode Developer Tools, and
         therefore it takes a lot of time to download and install.
   # Item It is similar to conda, pip, or apt and other GNU/Linux package managers, and we will use it in this project.

   # Now confirm you have the command line developer tools/
   $ xcode-select -p
   /Library/Developer/CommandLineTools
   ```

   Info: You will be prompted with a pop-up window to complete installation.

2. Mac Ports v2.8.1. This can also be installed on their website.

   - You will want to complete the following installation in your "Software/" directory, so that it is a part of your user path.
   - Find a folder to keep your software folder (e.g /Documents/RNAanalysis) and run:

     Zsh/Bash

     ```
     $ mkdir Software.
     $ cd Software.
     ```

Zsh/Bash

```
# Macports downloads as a zip file, so instead of saving the zip file to disk and then unziping it, you can pass
# the package directly to tar to unzip using the pipe operator "|"
$ sudo curl -L https://github.com/macports/macports-base/releases/download/v2.8.1/MacPorts-2.8.1.tar.gz | tar xj
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                              Dload  Upload   Total   Spent    Left  Speed
0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
1       00 21.4M  100 21.4M    0     0  3017k      0  0:00:07  0:00:07 --:--:-- 3201k

$ cd MacPorts-2.8.1/
$ ./configure CC=gcc # The out put of this is going to be a large list of checks and configuration commands and checks.
#if the configuration file ran correctly, the final line of the output should be:
config.status: creating src/config.h
# If you get a filepath/directory error, it may be because there are unescaped spaces in the file path.

$ make # Thsis will also give a large output with the last line being something like:
gzip -c portundocumented.7 > portundocumented.7.gz
* Warning: Using pre-generated man pages only.
* asciidoc, xsltproc (port libxslt) and docbook-xsl-nons are required to generate man pages from source.

$ sudo make install     # The last lines of output for this should be:
Congratulations, you have successfully installed the MacPorts system. To get the Portfiles and update the system, add /opt
      /local/bin to your PATH and run:

sudo port -v selfupdate

Please read "man␣port", the MacPorts guide at https://guide.macports.org/ and Wiki at https://trac.macports.org/ for full
      documentation.

Installing the "mpstats" port will enable submission of anonymous information about your system and installed ports to our
       database for statistical purposes: <https://ports.macports.org/>

# Finally, you can delete the installation directory.
$ cd ../
$ rm -r MacPorts-2.8.1
```

3. wget v1.21.4

Zsh/Bash

```
$ sudo port install wget
```

4. Java Development Kit 20 (jdk-20)

5. R v3.5.1

6. git v2.39.2

7. autoconf v2.7.1

4.1.2 Preprocessing Software

1. Zsh/Bash

```
$ cd Software/opt/bin/
$ git clone https://github.com/s-andrews/FastQC/
```

1. Trimmomatic version 0.39

2.

4.2 Building Using Conda

4.3 Installing a Pre-Built Conda Enviroment

5 Software Implementation

Algorithm 1: `FastTwoSum`

```
Input: $(a, b)$, two floating-point numbers
Result: $(c, d)$, such that $a + b = c + d$

if $|b| > |a|$ then
    exchange $a$ and $b$ ;
end
$c \leftarrow a + b$ ;
$z \leftarrow c - a$ ;
$d \leftarrow b - z$ ;
return $(c, d)$ ;
```

6 Pre-Processing Automation Using Bash Scripts

Creating a script:

- We first ned to create a bash file witht he extension ".sh". You can do this in any text editor but vim is a nice built in option. For details on how to use vim see appendix (?).

Zsh/Bash

```
$ vim organize_fastaQ_files.sh
```

- This will change your terminal to look like:

Vim

```
~
~
~
~
"format_fastaqs.sh" [new]
```

- We now need to tell the script which interpretor we would like to use (bash in this case, but you can use interpreters for other interpreted languages such as python or R.)
- To tell the script which interpreter we want to use we use a shebang, which is comprised of #! and the full path to the interpreter.

Vim

```
#! /bin/bash
~
~
~
~
~
"format_fastaqs.sh" 1L, 13B
```

Commands

- Commands in Bash Scripts are the same as those you would use in a Bash commandline. For instance mkdir makes a directory whether it's added to a script or used directly in the commandline.

Vim

```
#! /bin/bash
mkdir extracted_fastaqs # Makes a directory for us to place our fastaQ files in.
~
~
~
~
"format_fastaqs.sh" 1L, 13B
```

Variables

- In bash variable are typically declared in all caps and called with the "$" operator. For instance we could create a variable named FASTAQ_FP = ./FASTQ_Generation_2023-09-06_06_48_25Z-691456767/* and call it with $FASTAQ_FP.

Vim

```
#! /bin/bash
mkdir extracted_fastaqs # Makes a directory for us to place our fastaQ files in.
FASTAQ_FP=./FASTQ_Generation_2023-09-06_06_48_25Z-691456767/*
echo $FASTAQ_FP
~
~
"format_fastaqs.sh" 1L, 13B
```

Warning: Note that if you place spaces around the assignment operator "=", bash will interpret this as a command instead of a variable declaration.

- Now we can run the exit out of vim with by pressing the ESC key and entering ":wq" for write quite, and finally run the script:

Zsh/Bash

```
$ bash format_fastaqs.sh # Here we explicitly tell the commandline to use the bash interpreter, which we don't need to do.
./RNAanalysis/FASTQ_Generation_2023-09-06_06_48_25Z-691456767/*

# Because we added a shebang, we can instead use:
$ ./bash format_fastaqs.sh
zsh: permission denied: ./format_fastaqs.sh
# Note that we don't have permision to exuce the file. We can check permissions with ls -l.

$ ls -l
-rw-r--r--   1 alec  staff   154 Sep 19 12:57 format_fastaqs.sh
# The long version of ls shows us that the user only has read(r) and write(w) privilages. We can change that

$ chmod u+x format_fastaqs.sh # The argument "u+x" adds execution privilages to the user.
$ ls -l
-rwxr--r--   1 alec  staff   154 Sep 19 12:57 format_fastaqs.sh #The user now has read, write, and execution(x) privilages

$ ./format_fastaqs.sh
./RNAanalysis/FASTQ_Generation_2023-09-06_06_48_25Z-691456767/*
```

Warning: Be carefull with your use of chmod. If used incorrectly, it could allow anyone to execute files on your machine.

For Loops

- Looping in Bash is very similar to looping in other scripting languages.
- The main components are:
  1. "for" to initialize the for loop.
  2. "in" to define the iterable to iterate over.
  3. "do" to define the command to execute.
  4. "done" to close the for looop.
- For loops can also be nested as with other languages. In this example, we will us nested for loops to access each file within the subdirectories.

Vim

```
#! /bin/bash

FASTAQ_FP=./FASTQ_Generation_2023-09-06_06_48_25Z-691456767/*
for fp in $FASTAQ_FP; do
                for f in $fp/*; do
                                echo "$f"
                done
done
~
~
~
"format_fastaqs.sh" 8L, 144B
```

Zsh/Bash

```
$ ./format_fastaqs.sh
./FASTQ_Generation_2023-09-06_06_48_25Z-691456767/ATEY090523-NC_1_L001/NC_1_L001_R1_001.fastq.gz
./FASTQ_Generation_2023-09-06_06_48_25Z-691456767/ATEY090523-NC_1_L002/NC_1_L002_R1_001.fastq.gz
./FASTQ_Generation_2023-09-06_06_48_25Z-691456767/ATEY090523-NC_1_L003/NC_1_L003_R1_001.fastq.gz
./FASTQ_Generation_2023-09-06_06_48_25Z-691456767/ATEY090523-NC_1_L004/NC_1_L004_R1_001.fastq.gz
./FASTQ_Generation_2023-09-06_06_48_25Z-691456767/ATEY090523-NC_3_L001/NC_3_L001_R1_001.fastq.gz
...
```

Conditional Statements

- Similar to for loop, conditional statements in bash are akin to other scripting languages:
- They consist of:
  1. The "test" statement to initialize the conditional test.
  2. The "&&" operator to establish an action if the test returns true.
  3. The "||" operator to establish an action if the test returns false.
- In bash you can also use "[" and "]" inplace of test.

Vim

```
test -d "foo" && echo "foo␣is␣present" || echo "foo␣is␣not␣present"
[ -d "Latex_Files" ] && echo "Latex_Files␣is␣present" || echo "Latex_Files␣is␣not␣present"
~
~
~
~
"test.sh" 2L, 159B
```

Zsh/Bash

```
$./test.sh
foo is not present
Latex_Files is present
```

Putting it together

Now with a few extra commands (see Appendix(?)) we can write our final fastaq oragnization script.

organize_fastaqs.sh

```
[ -d "organized_fastaqs" ] && echo "Directory:␣organized_fastaqs␣exists" ||
   ↪ mkdir organized_fastaqs
FASTAQ_FP=./FASTQ_Generation_2023-09-06_06_48_25Z-691456767/*
for fp in $FASTAQ_FP; do
  for f in $fp/*; do
    base="$(basename␣$f)"
    echo "Copying:␣$base"
    [ -f ./organized_fastaqs/$base ] && echo "$base␣is␣already␣organized!"
       ↪ ||  cp $f ./organized_fastaqs/$base
  done
done
```

Zsh/Bash

```
$ chmod +x hello.py
$ ./hello.py

Hello World!
```

7 Defferential Expression Analysis using R or Python

8 References

References

[LD11] Bo Li and Colin N. Dewey. “RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome”. In: *BMC Bioinformatics* 12.1 (2011), p. 323. DOI: 10.1186/1471-2105-12-323. URL: https://doi.org/10.1186/1471-2105-12-323.

[BLU14] Anthony M Bolger, Marc Lohse, and Bjoern Usadel. “Trimmomatic: a flexible trimmer for Illumina sequence data.” eng. In: *Bioinformatics* 30.15 (Aug. 2014), pp. 2114–2120. ISSN: 1367-4811 (Electronic); 1367-4803 (Print); 1367-4803 (Linking). DOI: 10.1093/bioinformatics/btu170.

[PMK14] Rob Patro, Stephen M Mount, and Carl Kingsford. “Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms”. In: *Nature Biotechnology* 32.5 (2014), pp. 462–464. DOI: 10.1038/nbt.2862. URL: https://doi.org/10.1038/nbt.2862.

[Bra+16] Nicolas L Bray et al. “Near-optimal probabilistic RNA-seq quantification”. In: *Nature Biotechnology* 34.5 (2016), pp. 525–527. DOI: 10.1038/nbt.3519. URL: https://doi.org/10.1038/nbt.3519.

[Pat+17] Rob Patro et al. “Salmon provides fast and bias-aware quantification of transcript expression”. In: *Nature Methods* 14.4 (2017), pp. 417–419. DOI: 10.1038/nmeth.4197. URL: https://doi.org/10.1038/nmeth.4197.

[Zha+17] Chi Zhang et al. “Evaluation and comparison of computational tools for RNA-seq isoform quantification”. In: *BMC Genomics* 18.1 (2017), p. 583. DOI: 10.1186/s12864-017-4002-1. URL: https://doi.org/10.1186/s12864-017-4002-1.

A Description of FastQC