Author: Bernadette Johnson
Updated: February 2023

How to Install and Use Salmon

About this Protocol

This guide is for novice bioinformaticians users who would like to set-up and run Salmon for the first time. In this protocol Salmon will be used to map and quantify reads to a reference transcriptome. It is important that you use already trimmed RNA-seq reads for this protocol and that you have a reference transcriptome (genome should be well annotated). This protocol can be followed up with a differential expression analysis.

Users should always read the original manual when planning an analysis: https://salmon.readthedocs.io/en/latest/salmon.html# and https://buildmedia.readthedocs.org/media/pdf/salmon/stable/salmon.pdf

For more bioinformatics tutorials: bernadettebiology.weebly.com/protocols--tutorials.html

BLUF

Computer requirements and recommendations:

- A Linux subsystem, or a Linux virtual box
- Additional hard drive, other than your Local Disk (C:) drive, with ~5 GB of free space.
- Demands on RAM and CPU can be set low (but will have longer run time).

Total programs downloaded in this protocol:

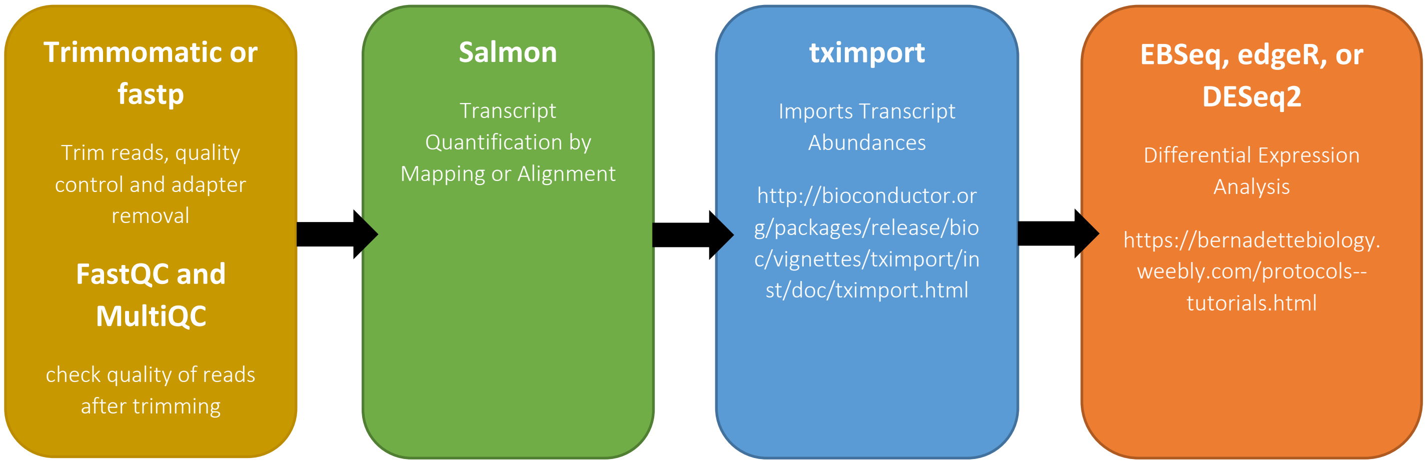
curl, zip, unzip, tar, autoconf, grep, cat, CMake, oneTBB, Boost, Salmon

Information for working example:

Context: For this project, I am interested in mapping trimmed, paired-end reads from a species of fish. Red text will indicate my specific path or file name.

System: I will be using the Ubuntu subsystem for on Windows 10 (Install Ubuntu on Windows).

General Suggested Pipeline


Trimmomatic or fastp
Trim reads, quality control and adapter removal
FastQC and MultiQC
check quality of reads after trimming
Salmon
Transcript Quantification by Mapping or Alignment
tximport
Imports Transcript Abundances
http://bioconductor.org/packages/release/bioc/vignettes/tximport/inst/doc/tximport.html
EBSeq, edgeR, or DESeq2
Differential Expression Analysis
https://bernadettebiology.weebly.com/protocols--tutorials.html

Let's get started

Update and upgrade

1. Update and upgrade your Ubuntu subsystem before starting. Open a new terminal window, then run the following commands. This will take a while if you have not updated and upgraded recently.

```
> sudo apt-get update
> sudo apt-get upgrade
```

Install prerequisites

2. Install these programs if they are not already installed. We will need these to install our other programs.

```
> sudo apt-get install curl
> sudo apt-get install libcurl4-openssl-dev  #curl library
> sudo apt-get install libssl-dev           #curl library
> sudo apt-get install zip
> sudo apt-get install unzip
> sudo apt-get install tar
> sudo apt-get install autoconf
> sudo apt-get install grep
> sudo apt-get install cat
#Alternatively, you can combine all these commands into:
> sudo apt-get install curl libcurl4-openssl-dev libssl-dev zip
unzip tar autoconf grep cat
```

CMake: Please note that using *sudo apt install cmake* will install an older version.

```
> sudo apt install -y software-properties-common lsb-release
> sudo apt clean
> wget -O - https://apt.kitware.com/keys/kitware-archive-
    latest.asc 2>/dev/null | gpg --dearmor - | sudo tee
    /etc/apt/trusted.gpg.d/kitware.gpg >/dev/null
> sudo apt-add-repository "deb https://apt.kitware.com/ubuntu/
    $(lsb_release -cs) main"
> sudo apt update
> sudo apt install kitware-archive-keyring
> sudo rm /etc/apt/trusted.gpg.d/kitware.gpg
> sudo apt update
> sudo apt install cmake

#This code is from user Teocci and more details to each step can
be found here: https://askubuntu.com/questions/355565/how-do-i-
install-the-latest-version-of-cmake-from-the-command-line
```

oneTBB:

```
> git clone https://github.com/oneapi-src/oneTBB.git
> cd oneTBB
> mkdir build
> cd build
> cmake -DCMAKE_INSTALL_PREFIX=/home/joneslab/onetbb
 -DTBB_TEST=OFF ..
> cmake --build .
> cmake --install .
```

Installing Salmon (Conda version)

3. Conda installation: Use this installation method if you already have a Conda environment (i.e., this: https://docs.conda.io/en/latest/) on your Linux subsystem. This will allow you to use Salmon only within the Conda (Python) environment. If you do not have Conda installed, follow the instructions below for the Linux version or Building from Source installation.

```
> conda update conda
> conda config --remove channels conda-forge
> conda config --add channels conda-forge

> conda install -c conda-forge -c bioconda salmon=1.9.0
#Make sure you specify the newest version of Salmon to install.

> conda activate    #This activates the Conda environment.
> salmon --version  #Check to make sure it properly installed.
> conda deactivate  #This deactivates the Conda environment.
```

Installing Salmon (Linux version, or Building from Source)

4. Linux installation: Download the latest version of Salmon. You can download it from the official site (https://github.com/COMBINE-lab/salmon). The latest version for me is v1.9.0. Please note that using *sudo apt install salmon* will install an older version. Do not use this version. Once it is done downloading, move the downloaded folder into the desired location or directory. Alternatively, you can use the command line to download the file into your working directory (this command is below).

```
> git clone https://github.com/COMBINE-lab/salmon.git
> cd salmon
```

5. There is another program that we need which is called Boost. We are going to install Boost using CMake, while we also install Salmon locally in the salmon directory.

```
> cmake -DFETCH_BOOST=TRUE
     -DTBB_INSTALL_DIR=/home/joneslab/onetbb
     -DCMAKE_INSTALL_PREFIX=/home/joneslab/salmon
     -DNO_IPO=TRUE
```

> make
#If this was successful you will see a message like: "[100%] Built target salmon". If it was **not successful** you will be presented with either (Error 1) a specific error code specifying a program that was not found or (Error 2) the error message "*/usr/bin/ld: error: lto-wrapper*".

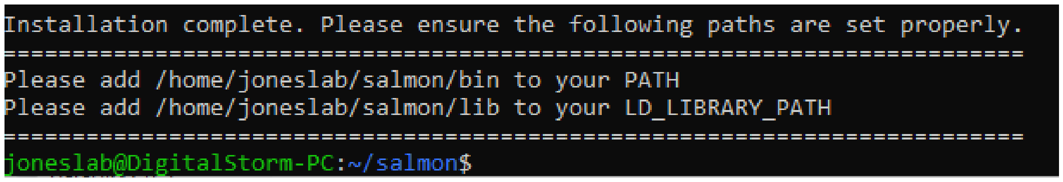
(Error 1) For the first error, you will have to read the error message carefully and perhaps Google to see what program you must install. For example, "could NOT find CURL (missing: CURL_LIBRARY CURL_INCLUDE_DIR)" would mean you would need to install CURL and its libraries using:

> *sudo apt-get install curl libcurl4-openssl-dev libssl-dev*

(Error 2) If you get a "*/usr/bin/ld: error: lto-wrapper*" failed message, you did not include the "*-DNO_IPO=TRUE*" flag in the *cmake* command.

When attempting to fix either error, you must (1) delete the CMakeCache.txt file, (2) install any programs that you might need, and (3) rerun the *cmake* command then the *make* command.

> make install
#If this was successful you will see a message like:



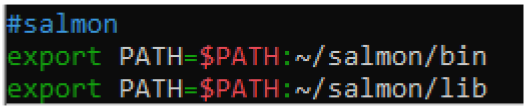
```
Installation complete. Please ensure the following paths are set properly.
==========================================================================
Please add /home/joneslab/salmon/bin to your PATH
Please add /home/joneslab/salmon/lib to your LD_LIBRARY_PATH
==========================================================================
joneslab@DigitalStorm-PC:~/salmon$
```

6. Now we will set the paths. This will make Salmon easily accessible, even when we are not working in the salmon folder.

> cd
> nano .bashrc

#Please do NOT change any other part of this file. Simply scroll down to the end of the file, append this to the end, and save:

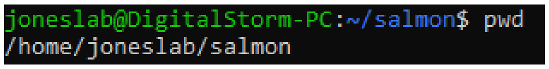
```
#salmon
export PATH=$PATH:~/salmon/bin
export PATH=$PATH:~/salmon/lib
```



```
#salmon
export PATH=$PATH:~/salmon/bin
export PATH=$PATH:~/salmon/lib
```

* This is my path to my folder. Your salmon folder might be in a different location. Whenever you want to check your current

PATH, or the location of a folder you can do one of two things: (1) you can navigate to the folder of interest and use the *pwd* command, (2) alternatively, you can navigate to the folder of interest and check the command line that lets you know where you are (the blue text). The first option gives you the full PATH name, the second starts from '~' which symbolizes your home directory.



```
joneslab@DigitalStorm-PC:~/salmon$ pwd
/home/joneslab/salmon
```

In this example, the path to my salmon folder can be written either as /home/joneslab/salmon or ~/joneslab/salmon.

> exit

#Re-open a new terminal.

> salmon --help

#If salmon is properly installed, you will see the usage and commands guide (see *Tips on installing programs* section for more information). Check to make sure the version number is up-to-date.

Before Running Salmon

7. For this project, I will be referencing the *Srivastava et al., 2020* Genome Biology paper. I will be running Salmon in mapping-based mode using their selective alignment method (SA).

   There are two options for implementing Salmon:

   (1) ***Mapping***: Use Salmon to map reads to a reference transcriptome, and then quantify reads.

   When to use: This option is for if you only have trimmed reads and have not previously aligned them.

   If you want this option: Follow the steps for ***Mapping-Based Mode*** in this protocol.

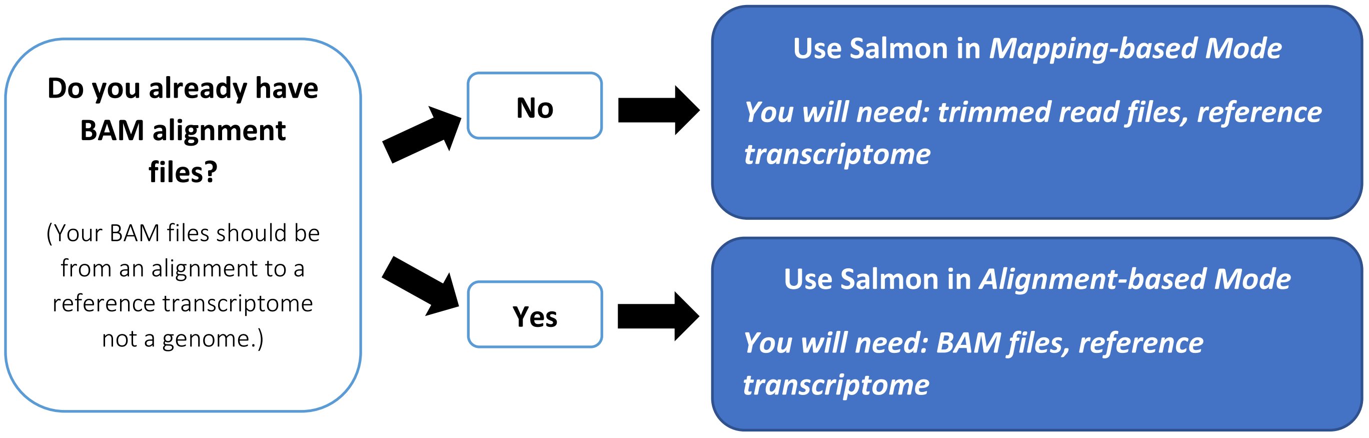
   (2) ***Alignment***: Import files from another mapping program and use Salmon only to quantify:

   When to use: This option is for if you already have alignment files that were produced from a separate program. These programs map reads to a reference transcriptome. BAM files aligned to a reference genome cannot be used here.

   If you want this option: If you already have your BAM files, then use Salmon on ***Alignment-based Mode***. This protocol is currently only for researchers who want to use Salmon in ***Mapping-Based Mode***. For this option, you will need to reference Salmon's original manual.

   Traditionally, Salmon has been used to only quantify reads (option 2). For this option, one would need to use a separate program that aligns reads to a reference transcriptome and produces alignment BAM files. These BAM files would then be imported into Salmon for read quantification. More recently, Salmon has updated their algorithms to include a useful mapping-based mode that promises to provide a quicker and computationally lighter option.

   Additionally, I am using trimmed, paired-end reads. This means I have used Trimmomatic to remove low quality and short reads, as well as Illumina adapters. It is important that you use trimmed reads. Installation and usage of Trimmomatic or other trimming programs are not included in this protocol.


Do you already have BAM alignment files?
(Your BAM files should be from an alignment to a reference transcriptome not a genome.)
No
Yes
Use Salmon in Mapping-based Mode
You will need: trimmed read files, reference transcriptome
Use Salmon in Alignment-based Mode
You will need: BAM files, reference transcriptome

Mapping-Based Mode Step 1: Running Salmon for Mapping

8. I will be used an official tutorial as a reference for this section. If you want to test this method out on mouse data, here is the link to the original protocol:

   https://combine-lab.github.io/alevin-tutorial/2019/selective-alignment/

9. Start by preparing your data. For Salmon to run with the selective alignment method, you need to create something called a "decoy" file. The file will contain a concatenated transcriptome and genome reference file, in that order. This file will be used to generate an index.

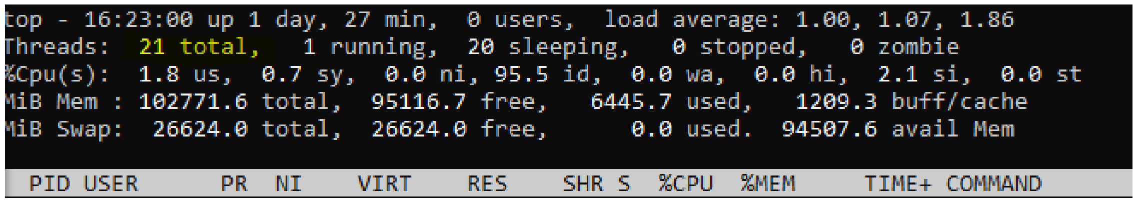
```
> grep "^>" <(gunzip -c pipefish_assembly.genome.fa.gz) | cut -d
" " -f 1 > decoys.txt
#We are using grep to find and pull out every header ">" in our
genome file. We then pipe "|" the output of our grep command
into our second command cut. This keeps the headers but removes
the DNA sequences.

#In this example, the pipefish genome is also compressed, and we
are using gunzip to unzip the genome and feed it into the
command, but to keep the original genome file compressed. This
will save some space on your computer. If you already have
uncompressed your genome file, then you can alternatively use:
     > grep "^>" pipefish_assembly.genome.fa | cut -d " " -f 1 >
     decoys.txt

> sed -i.bak -e 's/>//g' decoys.txt
#We are using sed to edit our newly create decoys text file and
remove the ">" part of the header.

To give an example, this is what the starting genome.fa.gz file
looks like:
>Chromosome_1 Syngnathus scovelli, whole genome sequence
cccatcatgctcgtttcttcgattcgatgctggcgctatatagtgcta
>Chromosome_2 Syngnathus scovelli, whole genome sequence
catgctcgtttcttcgattcgatgctggcgctatatagtgctacatac

This is an example of what the decoy text file would look like
after the cut command:
>Chromosome_1 Syngnathus scovelli, whole genome sequence
>Chromosome_2 Syngnathus scovelli, whole genome sequence

This is an example of what the decoy text file would look like
after the sed command:
Chromosome_1 Syngnathus scovelli, whole genome sequence
Chromosome_2 Syngnathus scovelli, whole genome sequence
```

```
> cat pipefish_transcriptome.fa.gz
pipefish_assembly.genome.fa.gz > gentrome.fa.gz
#Here we are using cat to combine the transcriptome and the
genome together into one file called gentrome. The .fa extension
indicates it is a fasta format style text file, and the .gz
means the files are compressed.
```

10. Generate an index. The index will help Salmon know the origin of a query sequence more efficiently. It saves computational power and time. Indexing will take a while depending on how large your genome is, and how much computational power (threads) you allow for. There are many options for running this command, so please check which options are best for your analysis.

```
> salmon index -t gentrome.fa.gz -d decoys.txt -p 16 -i
salmon_index
```

1. **-t** #This is an input file. It is a fasta file and contains the reference transcriptome and genome. The reference transcriptome sequences should appear first in the file, followed by the genome sequences.
2. **-d** #This is an input file. It contains the sequence ids from the reference genome.
3. **-p** #This is the number of threads you are allowing Salmon to use. The default is set to use 2 threads. If you can provide more than 2 threads, the process will be faster. Check the number of threads your computer has using the > *top -H* command. I have 21 total, but 1 is running so I have 20 available threads. I don't want to use all 20 because would like to run other scripts while this command runs, so I am only using 16.



```
top - 16:23:00 up 1 day, 27 min,  0 users,  load average: 1.00, 1.07, 1.86
Threads:  21 total,   1 running,  20 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.8 us,  0.7 sy,  0.0 ni, 95.5 id,  0.0 wa,  0.0 hi,  2.1 si,  0.0 st
MiB Mem : 102771.6 total,  95116.7 free,   6445.7 used,   1209.3 buff/cache
MiB Swap:  26624.0 total,  26624.0 free,      0.0 used.  94507.6 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
```

4. **-i** #This is an output index files.
5. **-k** #This indicates the minimum accepted length of a k-mer that will be used for the quasi-index. If you have shorter reads, you should have a smaller k. A k of 31 is recommended for 75 bp or longer reads, and is the default when k is not specified. (I used the default setting so I did not use this tag in my example).
6. **--gencode** #This flag is used when the input transcript fasta is in GENCODE format. It will split your transcript name at

```
the first "|" and this will be how your transcript names
appear on your output file. (My fasta file was not in gencode
format so I did not use this tag in the example.)
```

11. Check to make sure index has been built. The command should end with "done building index". Additionally, you can check for a new folder in your directory called "salmon_index".

```
> ls salmon_index   #Check for folder called salmon_index.
```

Mapping-Based Mode Step 2: Running Salmon for Quantification

12. Now we will quantify paired-end reads against the newly built index. There are over 50 different parameters for this command. Luckily a handful of these parameters are considered more important than others and I suggest to at least read about those important parameters from the Salmon manual. For this protocol, I will focus on the ones that I found to be most important for my research. In this example I have two samples: C4FL and C5FL. These samples are trimmed paired-end reads, the forward reads are labelled as "R1", and the reverse reads are labelled as "R2". You need to run this command for each individual sample, separately.

```
> salmon quant -i salmon_index -l A -1 C4FL_R1_paired.fastq -2
C4FL_R2_paired.fastq --validateMappings -o C4FL_transcript_quant
--gcBias --seqBias --softclip -p 18

> salmon quant -i salmon_index -l A -1 C5FL_R1_paired.fastq -2
C5FL_R2_paired.fastq --validateMappings -o C5FL_transcript_quant
--gcBias --seqBias --softclip -p 18
```

```
1. -i #This is an input. It is the name of the folder containing
   all of our index files.
2. -l #This is our library type. You can specify a specific
   library type (read Salmon manual) or you can ask Salmon to
   automatically detect this by using the "A" option.
3. -1 #This is a single forward read in fastq format.
4. -2 # This is a single reverse read in fastq format.
5. --validateMappings #This may be a default option depending on
   your version of Salmon (and so you may not need to actually
   include this tag), but it allows for selective alignment.
6. -o #This will be your output folder.
7. --gcBias #This helps to correct for GC bias in your reads. You
   can check for GC bias in your samples by running FASTQC on
   your trimmed reads. You can also run this option without
   checking for GC bias, as it will not impair the quantification
   of samples without GC bias. This is recommended if you plan to
   run a differential expression program.
```

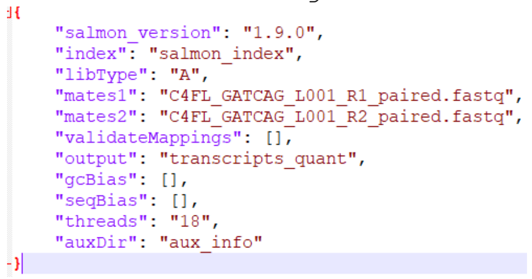
8. **--seqBias** #This helps correct for sequence-specific bias (specifically random hexamer primer bias). This bias is sometimes a product of Illumina sequencing, specifically during cDNA synthesis. This bias is a growing concern in transcriptome analyses. The method of correction has been improved and should not be prone to over-fitting if your data does not have a high level of this bias.
9. **--softclip** #This causes Salmon to ignore the 3' and 5' UTR regions of your reads. To understand why you would need this, please read *Salmon Warnings and Concerns:1.2*.
10. **-p** #This is the number of threads you are allowing Salmon to use. The default is set to use 12 threads. If you can provide more than 12 threads, the process will be faster. Check the number of threads your computer has using the *> top -H* command.

Mapping-Based Mode Step 3: Understanding Output

13. Running the salmon quant command should have generated a new output folder for each sample. For the example in this protocol this folder is called "sample_transcripts_quant". I will go over the most important output files here.

```
> ls C4FL_transcript_quant
```

1. **aux_info** #This contains files that report the amount of bias observed in your sample.
2. **cmd_info.json** #This is a record containing information on the parameters used to run Salmon. These are the parameters you gave to Salmon. This is important if you want to rerun Salmon consistently, or if you want to show others how you ran Salmon. It might look something like this:

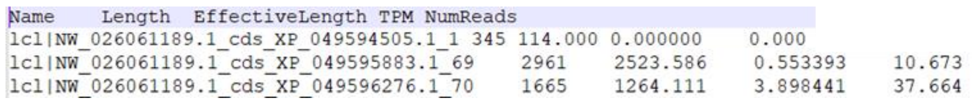


```
{
    "salmon_version": "1.9.0",
    "index": "salmon_index",
    "libType": "A",
    "mates1": "C4FL_GATCAG_L001_R1_paired.fastq",
    "mates2": "C4FL_GATCAG_L001_R2_paired.fastq",
    "validateMappings": [],
    "output": "transcripts_quant",
    "gcBias": [],
    "seqBias": [],
    "threads": "18",
    "auxDir": "aux_info"
}
```

3. **lib_format_counts.json** #This is a record containing some specific parameters used to run the quantification command. These parameters are usually what Salmon has inferred from the input files.
4. **logs** #Contains salmon_quant.log. This file provides important information on the number of fragments that mapped and the overall mapping rate. This is important to check after each

run to make sure there are not an unusual or unexpectedly large proportion of reads that did not properly map. To understand why you would need this, please read *Salmon Warnings and Concerns:1.2*.

5. **quant.sf** #This is the main output file. It contains (1) **Name:** the name of the target transcript from your input files, (2) **Length:** the length of the target transcript in nucleotides, (3) **EffectiveLength:** the effective length (when you remove things like GCbias, and other things), (4) **TPM:** this is the relative abundance of each transcript, and (5) **NumReads:** the number of reads mapping to each transcript.



```
Name    Length  EffectiveLength TPM NumReads
lcl|NW_026061189.1_cds_XP_049594505.1_1 345 114.000 0.000000    0.000
lcl|NW_026061189.1_cds_XP_049595883.1_69    2961    2523.586    0.553393    10.673
lcl|NW_026061189.1_cds_XP_049596276.1_70    1665    1264.111    3.898441    37.664
```

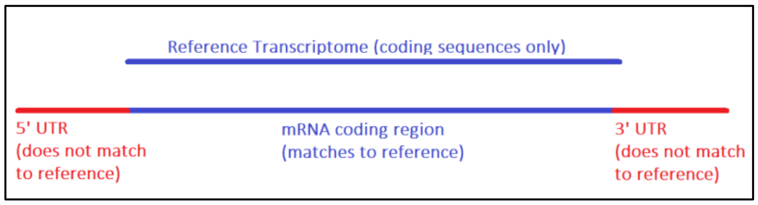
Salmon Warnings and Concerns

Issue 1. Mapping rate is lower than expected: Typically, you would expect to see a mapping rate of 80% or higher when mapping to a reference transcriptome. There are many things that can influence the mapping rate and having a lower mapping rate does not necessarily mean there is something wrong with your data. There are three main things to check when having a lower-than-expected mapping rate:

1.1. Did you trim your reads? The presence of adapters and low-quality reads can influence and lower your mapping rate. A good way to check if your reads are sufficiently trimmed is by running FASTQC on your files. Trim your reads using a program like Trimmomatic or fastp.

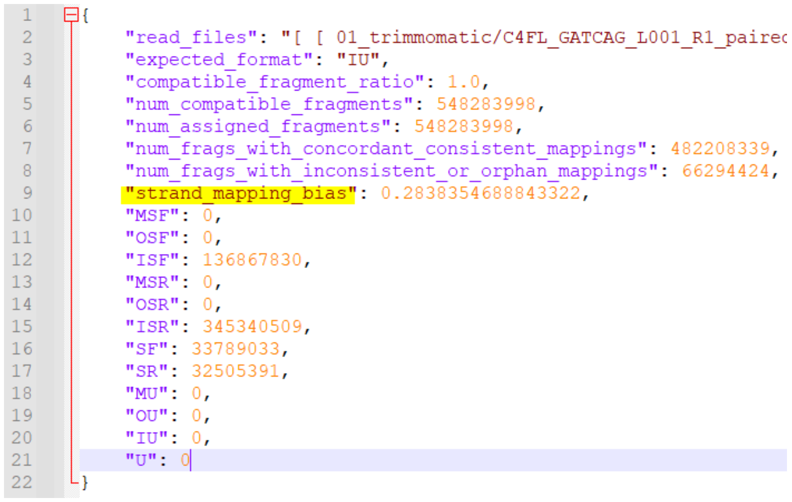
1.2. Are you mapping to an *annotated* transcriptome containing only coding sequences? A lower mapping rate might be expected when mapping to only the coding regions of a transcriptome. Usually your RNA-seq files will contain many different kinds of RNA. There will be both mRNA and various forms of non-coding RNA (rRNA, tRNA, miRNA, siRNA, and so on). mRNA codes for proteins, and this is what your annotated reference transcriptome is based on. Therefore, when you attempt to map a file containing multiple different RNAs to a reference containing only coding sequences you are only mapping a subset or your file, the portion containing the mRNA. For example, if you were mapping to an entire reference *genome* you would expect your mapping rate to be extremely high (around 90%). This is because you can map to non-coding regions. When mapping to an annotated transcriptome, having a mapping rate of around 50% is not uncommon.

If you plan to map to an annotated transcriptome, make sure that this annotation is fairly complete or "well-annotated". This can be determined by looking over an annotation report. If your organism has a genome available via NCBI, you can check the "Genome Assembly and Annotation report". Check the number of CDS (protein coding genes). Is it what you expect from your organism? Does that number align with the estimated genome size of the organism?


Reference Transcriptome (coding sequences only)
5' UTR (does not match to reference)
mRNA coding region (matches to reference)
3' UTR (does not match to reference)

Finally, your mRNA reads might have an untranslated region on both ends (5' and 3' UTR). These are not coding regions, and thus are not included in your reference CDS-only transcriptome. Salmon is fairly conservative when it attempts to find matches. For this reason, you should allow Salmon to ignore both of these ends that do not match up by using the --*softclip* option when quantifying.

<u>**Issue 2. Strand bias detected:**</u> If you see this warning: “Detected a *potential* strand bias > 1% in a unstranded protocol check the file: transcripts_quant/lib_format_counts.json”.



```
{
    "read_files": "[ [ 01_trimmomatic/C4FL_GATCAG_L001_R1_pairec
    "expected_format": "IU",
    "compatible_fragment_ratio": 1.0,
    "num_compatible_fragments": 548283998,
    "num_assigned_fragments": 548283998,
    "num_frags_with_concordant_consistent_mappings": 482208339,
    "num_frags_with_inconsistent_or_orphan_mappings": 66294424,
    "strand_mapping_bias": 0.2838354688843322,
    "MSF": 0,
    "OSF": 0,
    "ISF": 136867830,
    "MSR": 0,
    "OSR": 0,
    "ISR": 345340509,
    "SF": 33789033,
    "SR": 32505391,
    "MU": 0,
    "OU": 0,
    "IU": 0,
    "U": 0
}
```

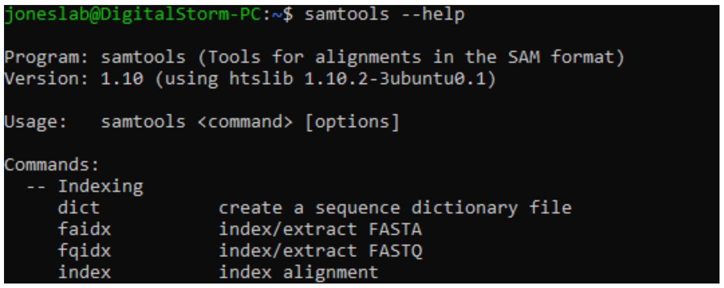
Start by opening the file *lib_format_counts.json* and find “strand_mapping_bias”. A perfectly unbiased ratio would be 50% or have strand mapping bias: 0.50. A warning is given with anything above 1%, so for example 0.52 would produce this warning. If your strand mapping bias is not too far off, you might be okay ignoring this warning. An extreme example would be like what is in the image 0.28 or 28%.

Strand mapping bias will be influenced by how you described the library type specifically the second part specifying if the protocol is stranded or unstranded. This depends on how you prepared the libraries for your samples. You might need to manually specify your library type instead of allowing Salmon to automatically determine this. It is also possible that one of your input files are not in the correct format and this file would need to be reformatted.

Tips on installing programs

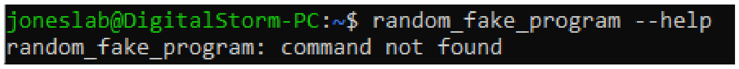
In this How-To document, we install several programs using the command line. A quick and easy way to check to see if a program is properly installed is to ask the program to display its usage and commands guide. For example, if we wanted to check our installation of a program called samtools we would use:

```
> samtools --help
```



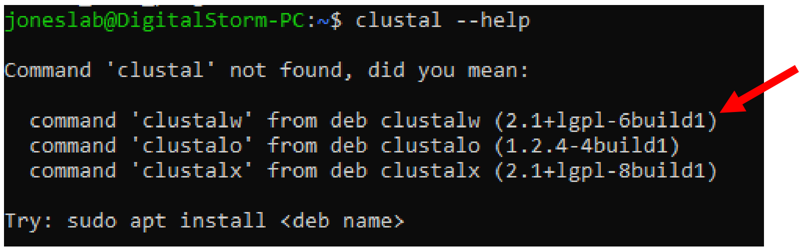
```
joneslab@DigitalStorm-PC:~$ samtools --help

Program: samtools (Tools for alignments in the SAM format)
Version: 1.10 (using htslib 1.10.2-3ubuntu0.1)

Usage:   samtools <command> [options]

Commands:
  -- Indexing
     dict           create a sequence dictionary file
     faidx          index/extract FASTA
     fqidx          index/extract FASTQ
     index          index alignment
```

If a program is not properly installed, you will instead see a message like this:



```
joneslab@DigitalStorm-PC:~$ random_fake_program --help
random_fake_program: command not found
```

If a program is not properly installed, but Linux knows a program with a similar name, you will instead see a message like this:



```
joneslab@DigitalStorm-PC:~$ clustal --help

Command 'clustal' not found, did you mean:

  command 'clustalw' from deb clustalw (2.1+lgpl-6build1)
  command 'clustalo' from deb clustalo (1.2.4-4build1)
  command 'clustalx' from deb clustalx (2.1+lgpl-8build1)

Try: sudo apt install <deb name>
```

If Linux is suggesting the correct program, then most times you can go ahead with its suggestion and install. You should note the version (arrow) and make sure that is the correct version you want to install. Linux might suggest an older version than what is currently available. If in this case I accept the suggestion, I would enter the following command:

```
> sudo apt install clustalw
```