Exercise 1: De novo transcriptome assembly using Trinity

Contact

Questions related to this exercise should be directed to Qi Sun (qs24@cornell.edu) and Robert Bukowski (bukowski@cornell.edu).

Data used in the exercise

RNA-Seq data used here, taken from Trinity workshop website (ftp://ftp.broadinstitute.org/pub/users/bhaas/rnaseq_workshop/rnaseq_workshop_2014/Trinity_workshop_activities.pdf), corresponds to *Schizosaccharomyces pombe* (fission yeast), involving paired-end 76 base strand-specific RNA-Seq reads corresponding to four samples: **Sp_log** (logarithmic growth), **Sp_plat** (plateau phase), **Sp_hs** (heat shock), and **Sp_ds** (diauxic shift). There are `left.fq` and `right.fq` FASTQ formatted Illlumina read files for each of the four samples. Although the reads represent genuine sequence data, they were artificially selected and organized so as to provide varied levels of expression in a very small data set, which could be processed and analyzed within the scope of a workshop.

Log in to your workshop machine

The machine allocations are listed on the workshop website: https://biohpc.cornell.edu/ww/machines.aspx?i=123.

Details of the login procedure using ssh or VNC clients are available in the document https://biohpc.cornell.edu/lab/doc/Remote_access.pdf.

Use your **ssh client** with BioHPC Lab credentials to open an **ssh session**. If you wish, you can open multiple sessions to have access to multiple terminal windows (useful for program monitoring).

Alternatively, use the **VNC client** to open a VNC graphical session (you will need to first start the VNC server on the machine from "**My Reservations**" page reachable from https://biohpc.cornell.edu after logging in to the website. To close the VNC connection, click on the "X" in top-right corner of the VNC window (but **DO NOT log out!**). This will ensure that your session (all windows, programs, etc.) will keep running so that you can come back to it by logging in again.

Prepare input files

If not yet done, create your subdirectory in the scratch file system **/workdir**. In the following, we will assume the user ID – please replace it with your own user ID.

```
cd /workdir
mkdir <yourID>
cd <yourID>
```

Copy the exercise files from the shared location to your scratch directory (it is essential that all calculations take place here):

```
cp /shared_data/Trinity_workshop_2018/part1/* .
```

When the copy operation completes, verify by listing the content of the current directory with the command `ls -al`. You should see 8 gzipped read files in a listing similar to this:

```
ls  -al
-rwxr-x--- 1 bukowski bukowski   868 Apr 3 13:31 my_trinity_script.sh
-rw-r----- 1 bukowski bukowski 5790168 Apr 3 13:31 Sp_ds.left.fq.gz
-rw-r----- 1 bukowski bukowski 5590326 Apr 3 13:31 Sp_ds.right.fq.gz
-rw-r----- 1 bukowski bukowski 5815390 Apr 3 13:31 Sp_hs.left.fq.gz
-rw-r----- 1 bukowski bukowski 5751383 Apr 3 13:31 Sp_hs.right.fq.gz
-rw-r----- 1 bukowski bukowski 2154125 Apr 3 13:31 Sp_log.left.fq.gz
-rw-r----- 1 bukowski bukowski 2097534 Apr 3 13:31 Sp_log.right.fq.gz
-rw-r----- 1 bukowski bukowski 5488286 Apr  3 13:31 Sp_plat.left.fq.gz
-rw-r----- 1 bukowski bukowski 5238362 Apr 3 13:31 Sp_plat.right.fq.gz
```

Along with the read files (`*.fq.gz`), a shell script `my_trinity_script.sh` containing the actual Trinity commands, is also provided for convenience.

Check the sequencing quality (optional)

This step summarizes the sequencing quality of the data. It is recommended to run this step before starting the assembly – it may help set the read trimming parameters for Trinity run.

```
cd /workdir/
mkdir qcreport
fastqc -o qcreport *.fq.gz >& qc_report.log &
```

- `-o qcreport` : specify the output directory (`./qcreport`) where the QC reports will be stored.
- All the fastq files should be specified, separated by space " ". The wildcard `*` also does the job.
- The last command will be run **in the background** (note the "`&`" at the end) and all its screen output will be saved in file `qc_report.log` (nothing special, but worth examining in case of run-time errors). The command should complete in about 15s (it takes longer with "real" data). When it does, hitting **"Enter"** will show the confirmation.
- After `fastqc` completes, you can use any **sftp** client (e.g., **FileZilla**) to copy the `qcreport` directory to you laptop computer, and open each of the `*_fastqc_report.html` files (one per each fastq file) with a web browser. If you are working in graphical environment (i.e., via VNC), you can launch the Firefox browser directly on the Linux workstation and navigate to `*_fastqc_report.html` files.

RNA-Seq data is expected to fail some of the tests run by the `fastqc` tool (higher than expected repetitious content, unequal nucleotide distribution in the beginning of a read due to the use of non-random primers) – this should not be a reason for concern. The `fastqc` results can be used primarily to decide the amount of sequence trimmed from each end of the read because of poor base quality.

Set up Trinity run

Although **Trinity** is launched with a single command, this command tends to be long and cumbersome to type. It is easier to include such a command in a shell script, where it can be easily examined and edited for future runs. A script like this, called `my_trinity_script.sh`, is provided for your convenience (it should have been copied to your scratch directory along with the input files).

Examine this script. While in the scratch directory `/workdir/`, open the file in a text editor (e.g., `nano` or `vim`) by typing

```
nano my_trinity_script.sh
```

The first line of the script tells Linux which interpreter to use to run the commands (here: **bash**). The second line defines a variable pointing to the directory where Trinity executable is located. This variable is then used (note "$" upfront) in the actual Trinity command, which occupies the subsequent lines of the script. Note that the "\" characters at line ends (they need to be the very last characters in line) serve the purpose of breaking long lines into readable pieces – otherwise the whole command would have to be written as a single line.

You may want to edit the options controlling the initial read trimming (see the relevant comment in the script) based on your analysis of the `fastqc` results (see previous section) – although the default parameters invoked implicitly with option `–trimmomatic` should be OK. You may want to add read normalization option (see the comment inside the script), although this will not have much effect with the limited data set used in his exercise. Do **not** change options `--CPU` and `--max_memory`. Since there are several users sharing each machine during the workshop, setting these options too high may cause the machine to run out of resources. The low CPU and memory settings proposed in the script are sufficient to complete the exercise. In the case of real run with real data, when the whole machine is dedicated to one Trinity instance, these options may and should be set much higher (see presentation for more hints).

After examining the script, exit the editor (**Ctrl-X** in `nano`). If you made any changes you want to keep, don't forget to save them upon exit.

Start Trinity run

Before starting Trinity, it will be convenient to open another terminal window – this will come useful later for monitoring the run. It you are accessing the machine using an **ssh client**, open another session by logging in again. If you use VNC connection, simply right-click anywhere with desktop and choose "Open in terminal" to open another terminal window.

In one of the windows, while in our scratch directory (if in doubt, enter `cd /workdir/`), make sure the script `my_trinity_script.sh` is executable, *i.e.*, there is an "**x**" in the 4th column when the file is listed with the `ls -al` commad:

```
ls -al my_trinity_script.sh
-rwxr----- 1 bukowski bukowski 695 Mar 5 13:56 my_trinity_script.sh
```

If needed, make the file executable:

```
chmod u+x my_trinity_script.sh
```

Then launch the script:

```
nohup ./my_trinity_script.sh >& my_trinity_script.log &
```

All screen output (info messages and error messages, if any) will be saved in the file `my_trinity_script.log`. The script will start executing in the background (the **&** at the end), so that the terminal will return to the prompt right after you hit "Enter". You can use it (together with the additional terminal you opened before launching Trinity) to monitor the run. Or you can log out (close the session) – the program will keep running. You can examine the results when you log in to the machine again.

The exercise run is expected to take a few minutes.

Monitor the Trinity run

There are several ways to see how a Trinity run is progressing:

Use the top command. In one of the terminal windows, run

`top -u <yourID>`

You will see a dynamically updated list of your processes, with the ones taking the most CPU on top of the list. You can also see the % of memory taken by each process. As Trinity progresses, you will see different program names on top of the list (e.g., `jellyfish, inchworm, bowtie2, samtools, salmon, GraphFromFasta, ReadsToTranscripts, perl, java`). Some of these programs are multi-threaded and will be shown as consuming about 200% CPU (corresponding to the `--CPU 2` setting). Others (like some perl scripts or java VM running Butterfly) will show as single-threaded processes running in parallel *(i.e.*, two processes, each consuming about 100% CPU). You may keep the **top** display running in one of the windows, or exit by hitting "**q**".

Peek into the log file. The screen output (here: saved into the file `my_trinity_script.log`) contains messages from the Trinity script itself as well as from the programs it calls. Although the messages may sound cryptic at times, they generally allow the user to figure out which stage of the calculation is running at the moment. It also contains useful timing information (start and end dates of individual stages). To look into the log file, you can use any of the following commands

`more my_trinity_script.log` (page through the file from the beginning)

`tail -100 my_trinity_script.log` (display the last 100 lines of the file)

`tail -f my_trinity_script.log` (continuously display incoming lines)

Of course, you can also look at the whole file by opening it in a text editor. Upon exit, **discard any changes** you may have inadvertently made.

Look into the output directory. As the run progresses, various intermediate files and directories are being produced in the output directory specified in Trinity command line using `--output` option (here: `/workdir//trinity_out`). First, if `--trimmomatic` option was invoked, the read cleanup will be run and cleaned read files will be written. Otherwise, if the input files were compressed with `gzip` (as in this example), Trinity will un-compress them. The FASTQ files (original or trimmed) will be then converted into FASTA format and combined into a single `both.fa` file located in the output directory. If the in-silico normalization was not suppressed, a subfolder `insilico_read_normalization` will be created in the output directory to store the normalization results. Most files in the output directory are named after the Trinity stage that produced them. In particular, files with names ending with `.ok` or `.finished` indicate that a given stage has successfully completed. The presence of the file `recursive_trinity.cmds` indicates that the run is in the final stage which involves processing of multiple independent assembly commands. This stage benefits the most from parallel processing on multiple CPU cores.

Restarting Trinity

Should a Trinity run fail for any reason, it can be re-started from the last successfully completed stage using the same command, possibly changed to correct for the reason of the crash (inferred from error messages in the screen log file, for example). Typically, crashes happen due to insufficient memory. The final stage of the run (Butterfly) is most susceptible to crashes. Restart with reduced `--CPU` setting will usually allow Trinity to run to completion.

Check the final result

Upon successful completion of Trinity, the assembled transcriptome is written to the FASTA file called `Trinity.fasta` located in the output directory (here: `/workdir/trinity_out`). If this file is not present, it means that Trinity did not yet finish (the top listing will then still be showing Trinity-related commands running), or that it crashed. In such a case, examine the log file for errors. For a quick check, execute (while in the scratch directory `/workdir/`)

```
grep -i error my_trinity_script.log
```

If the above command does not produce any output, the run went smoothly.

`Trinity.fasta` contains transcripts to be evaluated, annotated, and used in downstream analysis of expression. In this exercise, we only concentrate on basis quality evaluation of the assembled transcriptome.

Basic contig statistics

Basic contig statistics can be obtained using a Trinity utility script `TrinityStats.pl`. In your scratch directory, type (on a single line)

```
    /programs/trinityrnaseq-v2.8.6/util/TrinityStats.pl
./trinity_out/Trinity.fasta
```

The output (written to the screen) will contain basic information about contig length distributions, based on all transcripts and only on the longest isoform per gene. Besides average and median contig lengths, also given are quantities **N10** through **N50**. **Nx** is the smallest contig length such that x% of all assembled bases are in contigs longer than **Nx**. Specifically, **N50** is the contig length such that half of all assembly sequence is contained in contigs longer than that. In whole genome assembly, N50 is often used as a measure (one of many) of assembly quality, since the longer the contigs, the better the assembly. In the case of transcriptome, contig lengths should be correct, which does not imply “large”. If it falls in the right ballpark (about 1000-1,500), N50 can still be used as a check on overall “sanity” of the transcriptome assembly.