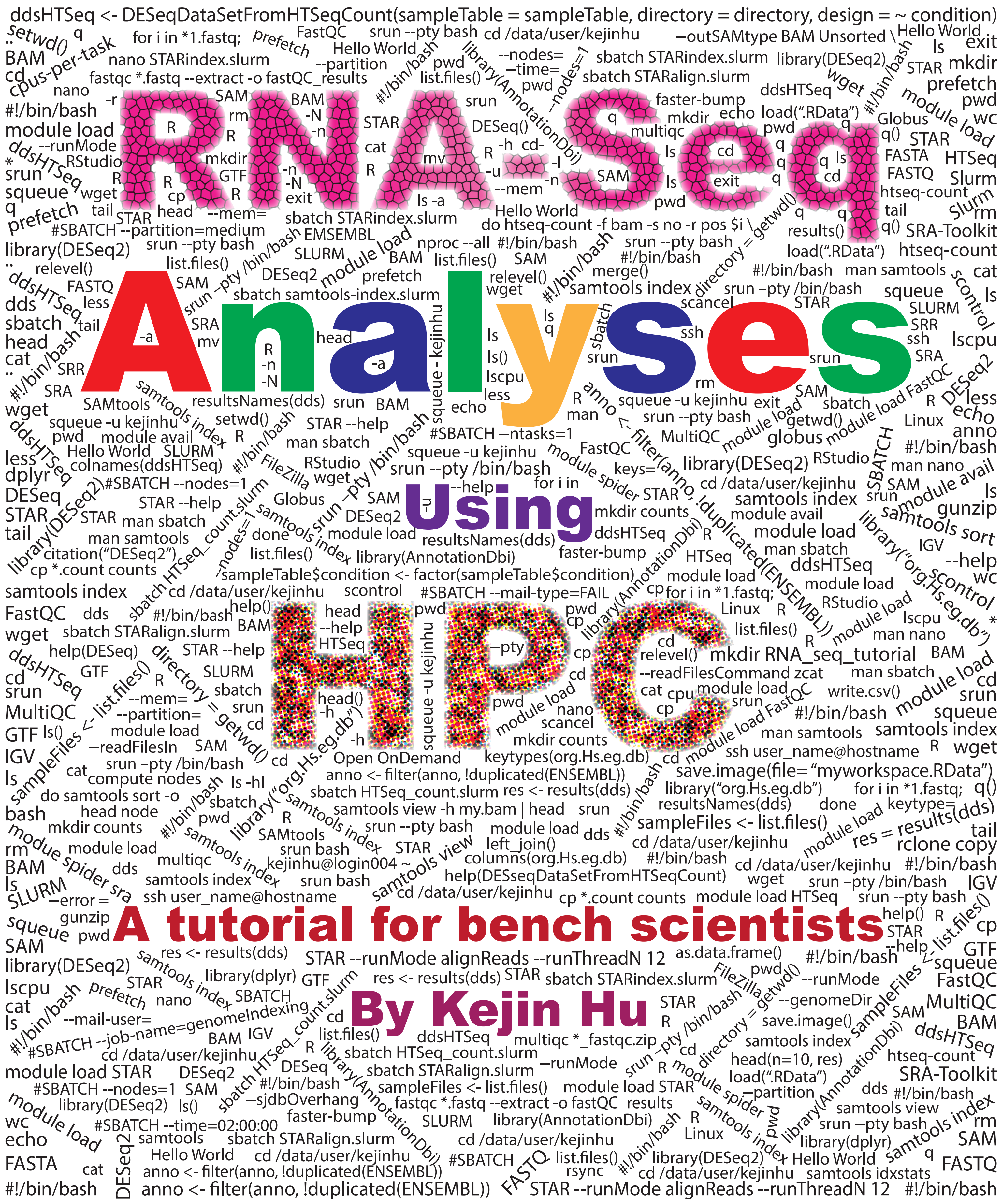


# RNA-Seq Analyses Using HPC

## A tutorial for bench scientists

By Kejin Hu

# Contents

# List of Figures

## About the cover

The background of the book cover is a word cloud of random bash/Linux commands, R functions, software names, snippets of bash/Linux or R scripts, which have been used in this tutorial. The four colors in the word of “analysis” represent the four bases in RNA.

# Preface

I am a cell biologist leading a small research lab in stem cell biology and cellular reprogramming. As a PhD student back from 1999 to 2003, I investigated gene expression by Northern blot analyses, and reverse transcription PCR (RT-PCR), which are tedious but still provide very little information from the perspective of current technologies. Nowadays, RNA sequencing (RNA-seq) is a commonplace experimental tool for biological and biomedical sciences. My lab is of no exception. I started to use RNA-seq technology after I established my own research laboratory in 2011. Amazed by the astronomical and global information RNA-seq can provide, yet I was stranded in analyzing the RNA-seq data of daunting size. Our institutional bioinformaticians gave me critical support in the analyses of our RNA-seq data during those early years. However, as a scientist I had an uncomfortable feeling of "being in the dark" when I did not understand how those data are processed and analyzed. In addition, when I have hundreds of gigabytes of data, I was not satisfied in that I need to wait for another 10 days or longer to see the statistical and normalized count data. Furthermore, the bioinformaticians generally conduct standard analyses and customized or project-specific analyses are not easy to achieve when you rely on someone else who is not quite a former member of your research team. This motivated me to learn Linux, R, high-performance computing (HPC), and many packages required for RNA-seq analyses. It was not an easy journey for a scientist without a degree in computer science, data science, or statistics. There were so much reading, googling, practicing, frustration, and email communications with experts. I realize now that an end-to-end accessible tutorial will save months of time for a bench scientist during this learning process. It took me around one year to become proficient in RNA-seq analyses after starting this adventure. I believe with this tutorial you may need just one month or even one week to become competent in RNA-seq analyses. It may reduce the situations of frustration, googling, and asking around. Given the pain I experienced in learning RNA-seq analyses, I would like to make this tutorial available to my fellow scientists, postdoctoral fellows, PhD students, and other lab workers. I hope this tutorial will quickly enable and empower you. I dedicate this volume to the experimental scientists in biological and biomedical sciences like me. Unlike the "abstract" tutorials from bioinformatics tool developers that make you feel lost, this is a "foolproof" tutorial from a bench scientist for bench scientists.

## Copyright statement

## Summary

RNA-Seq analysis software has been well-established and mature. High-performance computing (HPC) has become available to the general science community, and this makes RNA-seq analyses much easier. Scientists in biological and biomedical sciences now can and should analyze RNA-seq data by themselves as in the case they use Excel, Illustrator, PowerPoint, and the specialized software in their respective fields on a daily base. With the infrastructure established, what the bench scientists need is an integrated, complete, and enabling tutorial. The current tutorial serves this purpose and provides a complete set of skills from end to end for RNA-seq analyses. The major packages introduced here are FastQC and MultiQC for quality control, STAR for indexing of a reference genome and sequence alignments, SAMtools for sorting, indexing and reviewing of the BAM/SAM file data, IGV for viewing the resulting alignments, HTSeq for counting reads of each feature, and DESeq2 for analyses of the differential expressions on the RStudio platform, all of which are widely used software in the RNA-seq field. The introduced skills include but are not limited to uploading of raw RNA-seq data to HPC, management of the data, downloading of the reference genome sequences and their annotations from the public database, generation of the reference genome indices, quality controls of the sequencing data, alignment/mapping of reads to the reference genome, sorting and indexing of the BAM file data, counting of reads to the features, review of data in various formats at each major step, differential expression analyses, combining of the result and read count tables, annotation of features in the result data frame, saving of the resulting data, and transfer of data to local storage from HPC. With the novices in mind, the very basic Linux commands are introduced in the context of RNA-seq analyses. It also introduces many basic skills about use of the HPC platform such as module usages, resource request using SLURM, and file managements. Taking advantage of HPC, this tutorial introduces the optimized procedures for the entire pipeline, which can significantly reduce the pipeline time so that one can complete analyses in less than one day for a typical RNA-seq project of most labs.  This is achieved by multiple threading at the data levels (i.e., simultaneous calculation/processing of data of one individual sample) and parallel calculation at the sample levels (i.e., simultaneous calculation/processing of many samples) at different major steps using various skills.

## Keywords

RNA-seq, RNA-seq analysis tutorial, STAR, HTSeq, DESeq2, FastQC, MultiQC, SAMtools, high-performance computing, SLURM, HPC, nano, Linux, R, RStudio, SRA-Toolkit, IGV, FASTQ, BAM, Globus, FileZilla

## Limitations

This tutorial aims to empower and motivate newbies of RNA-seq analyses. As a primer to RNA-seq analyses, it provides the basic procedure of a selected pipeline so that the audiences can independently conduct the end-to-end procedure, but it does not provide comprehensive procedures for any software introduced here. For complete software documentation, extensive tutorials and advanced skills, audiences should consult the original literature and individual tutorials of each software (SRA-Toolkit, STAR, HTSeq, FastQC, MultiQC, IGV, nano, SLURM, DESeq2, Linux, R, RStudio, and others). There are other pipelines, but this tutorial uses the popular STAR for alignment, HTSeq for counting of reads, and the widely used DESeq2 for analyses of differential expressions. After you become proficient with one pipeline it is not difficult to establish another pipeline by your own.

## Acknowledgement

I am indebted to the UAB HPC team, who provided a lot of support to me via Zoom during the virtual office hours when I initially needed assistance using HPC. I dedicate this book to my wife, Zhihong Xu, and my teenage son, Drakeson Hu, who are the motivation for my everyday work. I thank Dr. Da Yan, an assistant professor in the Department of Computer Science, UAB, who always replied to my emails in a timely manner with answers to my questions when I converted from Windows to Ubuntu and analyzed my RNA-seq data on a surplus PC. I also give my special thanks to Drs. Tim Townes, Louise Chow and Igor Slukvin who have always been supportive of my efforts and ambitions. I am very grateful to the Linux, R, and bioinformatics communities; there are always answers to my questions out there on the internet because of these outstanding communities. Finally, I greatly appreciate the friendship of Dr. Kevin Pawlik. Science is fun. Being in academia is challenging. Every long day of work, taking a break to chat with Kevin is like a cup of coffee for coffee drinkers.

# Chapter 1

## Introduction

RNA sequencing (RNA-seq) has become a commonplace tool in biological and biomedical sciences. The cost of an RNA-seq run is at the level of or less than that of a vial of a common antibody but the information from an RNA-seq experiment is astronomical and global compared to that of a western blot or an RT-PCR experiment. To most laboratories, the bottleneck is at the stage of data analyses, and bench scientists heavily rely on the full-time professional bioinformaticians. Bioinformatics service is frequently not available to many underprivileged laboratories especially in the current funding situation. Even for the research groups with bioinformatics support, the ability for a bench scientist to analyze RNA-seq data has much benefit and can make a difference in scientific discoveries. RNA-seq analysis tool/software has now become established and mature. Progress in infrastructure in both software and hardware development now makes it possible that a bench scientist can readily analyze RNA-seq data given an accessible, enabling, and end-to-end tutorial.

RNA-seq analyses generally require skills in Linux and R programming. This prerequisite discourages many bench scientists to analyze RNA-seq data. Learning Linux command lines seems difficult, but I agree with the statement that "it's not that it's so hard, but rather it's so vast" [1]. The fact is that Linux is easy! It just has too many Linux commands. The good thing is that we do not need a lot of commands to conduct RNA-seq analyses. A survey by the Standish Group indicates that less than 5% of the functionalities of any application software are generally used by an ordinary user [2]. I regularly use Word, Excel, PowerPoint, Illustrator and Photoshop, but I may just use less than 5% of their functionalities. We just need a small fraction of the command reservoir to conduct RNA-seq analyses although it involves two major platforms (Linux and R platforms) and many Linux modules and R packages. This tutorial will introduce these <5% of the commands in those packages needed to analyze RNA-seq data efficiently and professionally. Of course, you could become an advanced user of Linux after you are on board. The most difficult time is at the beginning when you adventure to a new area.

RNA-seq analyses generally need at least a high-end desktop computer, a workstation, or a server because of very high requirement on computational resources. For example, the popular and fast aligner STAR requires at least 32 gigabytes (G) of RAM for RNA-seq alignments to the human genome [3, 4]. Almost all research universities now have or have access to a high-performance computing (HPC) or supercomputing facility. With HPC we usually do not worry about the memory and storage, which are the two major limiting factors in using a desktop computer for bioinformatics. RNA-seq analyses need Linux system and many Linux-oriented software. Even though you have a decent desktop or workstation the installation and configuration of various specialized software are a headache for bench scientists since you need to use command lines to install and configure the specialized software. At the beginning, I took a surplus PC with 32 G of RAM, replaced its Windows with Ubuntu, and installed the specialized software for RNA-seq analyses. I nearly gave up at this frustrating stage. With HPC, you do not need to install the software. Many widely used software modules have already been installed on the clusters by your HPC team, and you just load the required software to use. Additional advantage is that we can easily work anywhere with HPC, which is difficult to achieve with your desktop computer or workstation. Last, more HPC facilities are adopting the web interface, and this reduces the barrier to HPC access new users usually encounter with the traditional SSH (secure shell) interface and command lines needed to conduct basic communication with HPC. Even without web interface, using HPC is not arcane, and in fact HPC is very simple to use. We just need the basic Linux skills, and skills about job submission and the specialized software for your analyses.

RNA-seq analyses require many Linux and R software packages, and the documentations for the software are highly specialized but are not very helpful to most biological and biomedical students/scientists. In addition, those documentations are isolated and package specific. With the bench scientists in mind, I thus provide notes even for many basic Linux and R commands. I truly believe one more page may not cost a dime in this era of digital storage, but it will save an audience hours of time and reduce googling and frustration, and even keep the audiences from dropping in this usually long and frustrating learning process. However, the audiences need some basic knowledge about Linux and R platform. For those without any prior Linux experience, I was one of you. I started with the book of “Learn Linux in 1 Day” by Krishna Rungta, “Learn Linux Quickly” by Ahmed Alkabary, and the more scholarly “The Linux Command Line” by William Shotts [1]. Readers can also refer to the Linux primer: “Essential Linux” as Appendix B in the textbook of “High-performance Computing – Modern Systems and Practices” [5]. A great news is that many essential skills mentioned in the primers are in fact not necessary for RNA-seq analyses. R is even easier with the availability of RStudio. I have provided brief introductions to R in two other tutorials [6, 7]. I learned R by reading and practicing the books of “The undergraduate Guide to R” [8] and “An introduction to R” [9].

Different sets of software can be used for RNA-seq analyses, and this tutorial uses the popular and ultra-fast STAR for alignment, the widely used HTSeq for counting, and the trusted DESeq2 for analyses of differential expression of genes. It employs the widely used SLURM (Simple Linux Utility for Resource Management) for HPC resource managements (https://slurm.schedmd.com/ ).

Figure 1 provides the workflow of RNA-seq analyses introduced in this tutorial. In this tutorial, the Linux commands, R functions and scripts will be indicated by red text. The notes and comments after scripts will be indicated by a pound sign #. Many supporting commands are included in the comment parts after the major commands/scripts. The scripts and comments about them are also distinguished by the DengXian Light font of a smaller size than the Times New Roman font used in the main text. Some screen outputs are included as blue texts, but the outputs of a command or computational task are usually not included because of space consideration.

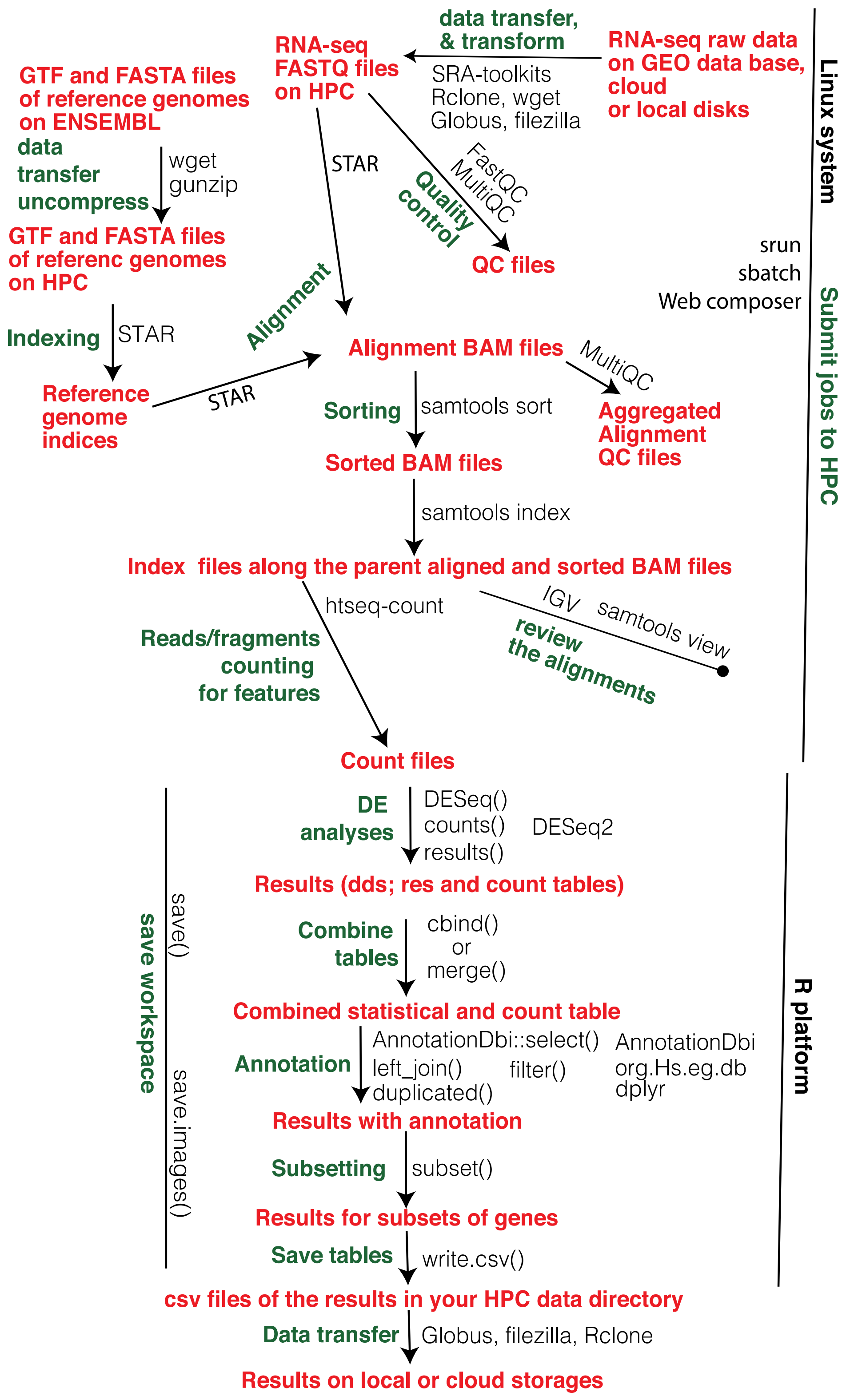


**Figure 1.** Workflow of RNA-seq analyses. Green texts by arrows indicate processes/procedures; red texts, data/file statuses; black texts by arrows, package/software or command names.

An RNA-seq project involves multiple samples and processing of individual samples using various software at different steps requires a lot of hands-on time. Serial processing of all the samples using one script with the bash *for* loop significantly decreases the hands-on time. This introduced strategy, however, does not reduce the machine time although it releases you from tedious typing. I addition, serial calculation may not make good use of the HPC resources and compute power. This tutorial therefore has also provided an optimized pipeline so that analyses of a typical RNA-seq project can be completed within one day. Taking advantage of the computation power of HPC, the fast pipeline is achieved by use of multiple threading at the data and sample levels, and parallel processing at the sample levels at various major steps. The following table depicts the differences for the optimized and non-optimized procedures.

**Regular and optimized computing time for the major steps**

| **Steps** | **Optimized (minutes)** | **Regular/not optimized** |
|---|---|---|
| Downloading RNA-seq data | 6 | 36 minutes |
| QC the 14 FASTQ files | 3 | 28 minutes |
| Indexing human genome | 21 | 3 hours 13 minutes |
| Alignment of 7 RNA-seq(paired-end) | 10 | 30 minutes to > 4 hours |
| Sorting of the 7 BAM files | 5 | 105 minutes |
| Indexing of the 7 BAM files | 1 | 7 minutes |
| Counting of the 7 samples | 84 | > 8 hours |
| **Total computing time** | **130 minutes** | **> 14 hours** |

**Notes**:

1) The computing time for DESeq2 steps are not included since the machine running time is not a limiting factor and the hands-on time dictates the total job time.
2) The above times are based on 7 human RNA-seq samples of paired-end sequencing with reads/sample ranging from 28 to 48 million and total reads of 280.4 million (average 40.1 million/sample), which are the typical sizes (for both sample number and sequencing depth) of an RNA-seq project for most labs.
3) The speed optimization is achieved by multiple threading at the data levels or parallelism at the sample level, or both, as well as use of the ultrafast aligner STAR.
4) The times at each major step above were the test results on the Cheaha clusters.

## Platform and test data

This tutorial was written in Microsoft Word for Mac version 16.63.1 on the macOS Big Sur version 11.3.1 with additional use of Adobe Illustrator and the screenshot function of iMAC. The RNA-seq pipeline was established and tested on Cheaha, which runs Red Hat Enterprise Linux (version 7.9) with the Open OnDemand portal. The DESeq analysis step was carried out on the RStudio server of Cheaha. This tutorial uses 7 published human RNA-seq samples of two cell types with an average sequencing depth of 40 million/sample, which are available from the GEO database.

# Chapter 2

# Work on HPC

# and

# Transfer RNA-seq FASTQ data onto HPC

FASTQ files are generally what a bioinformatician starts with in his/her analyses of RNA-seq data. Your raw RNA-seq data in the FASTQ format may sit in a sequencing facility, a database, or your local storage. This chapter therefore introduces different methods for transferring your raw RNA-seq data onto HPC for analyses.

## 2.1 Logon your HPC account and generate project directories on HPC

Nowadays, supercomputers usually have or will have an intuitive and easy-to-use web portal. This tutorial thus introduces access to HPC via a web portal powered by Open OnDemand developed at the Ohio Supercomputer Center (OSC) [10], on a browser. The access to HPC via SSH terminal is also introduced but it is more "exotic" to the biological and biomedical scientists/students. Open OnDemand has been used by many universities including Harvard, Yale, Stanford, Princeton, Caltech, and others. Your HPC web portal may have a different name (e.g., HPC web interface, HPC gateway, HPC board, or others) and appears different, but upon login you will use almost the same codes/commands introduced in this tutorial.

You need a HPC account to access your HPC. Login to your HPC account via the web portal (at rc.uab.edu in the case of Cheaha on the campus of University of Alabama at Birmingham, UAB) using your credentials. If you do not have an account yet, contact your HPC team to setup your account. After login, you can go to the HPC shell terminal by clicking the ">_Cluster Shell Access" in the pulldown menu under the Clusters tab on the HPC web dashboard (Cheaha in this tutorial) (Figure 2). This will take you to your home directory on the HPC login nodes as indicated by a tilde ~ sign (Figure 3). The login node is indicated by UserName@loginNNN (underlined in red in Figures 3, 5 and other screenshots).

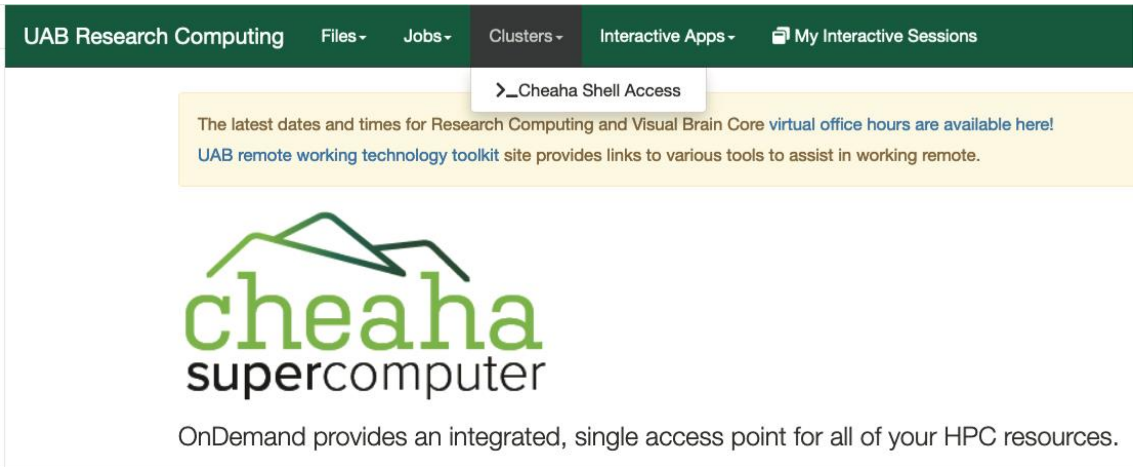
HPC may use different Linux distributions. If you are curious about which Linux your HPC is using you can find out by issuing the following command after the dollar prompt sign $ in the shell terminal,

```
lsb_release -a            # Cheaha uses Red Hat Enterprise Linux. Your HPC may use
Debian, Ubuntu, or others. Alternatively, you can find similar information by calling
cat /etc/*release.
```

You can find out the full path to your home directory using the pwd (Print Working Directory) command after the $ prompt sign (the first command in Figures 3 and 5),

```
pwd     # This outputs the full path to your working directory on your terminal screen.
```

If you are curious about what your HPC login nodes look like, you can see the specifications of your login nodes by calling the display/list CPU command lscpu (the second command in Figure 3), lscpu # The output of this command gives information about the login node CPUs (The contents between commands lscpu and ls in Figure 3).



The latest dates and times for Research Computing and Visual Brain Core virtual office hours are available here!
UAB remote working technology toolkit site provides links to various tools to assist in working remote.

OnDemand provides an integrated, single access point for all of your HPC resources.

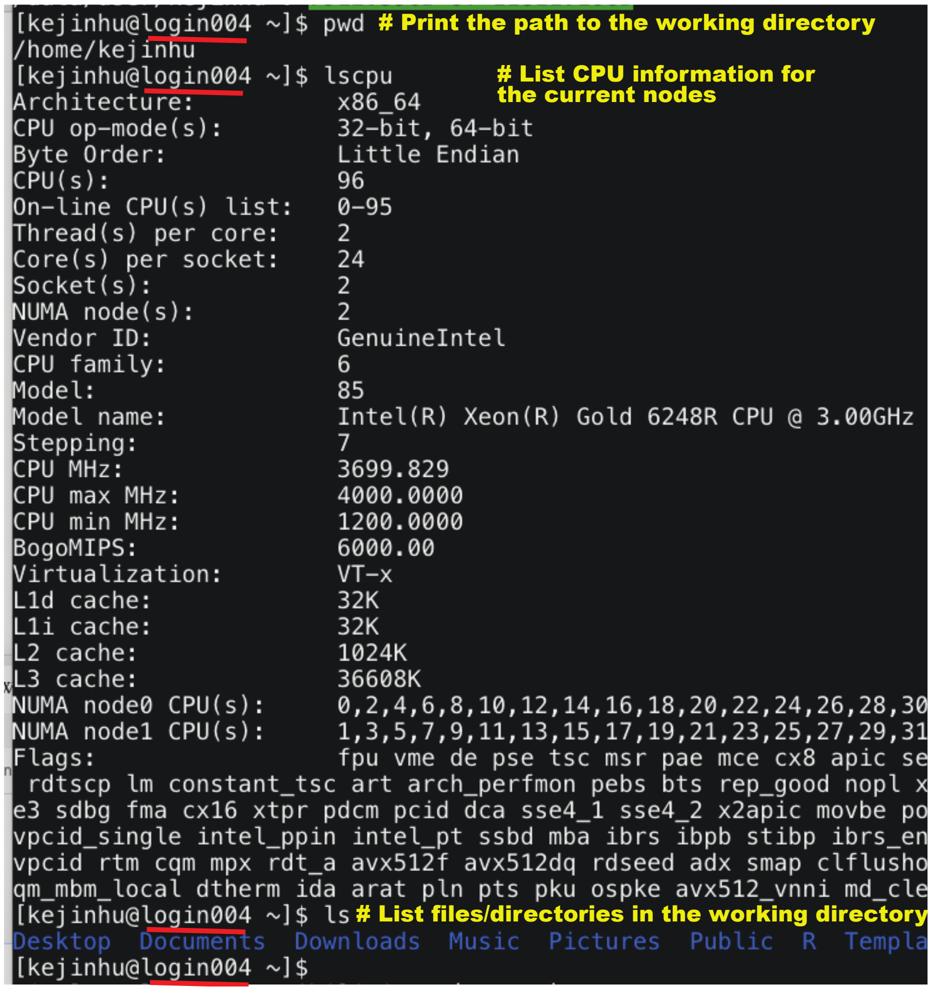
**Figure 2**. HPC web interface with Cheaha as an example. The pulldown menu for Clusters (the grey tab) is extended and the Shell Access menu is exposed in this screenshot.

To see the subdirectories/files on your home directory, issue the list command ls (the last command in Figures 3 and 5),

ls # The ls command here outputs a list of subdirectories of my home directory /home/kejinhu, which are in blue text (bottom in Figures 3 and 5), and files (black in Figure 5, i.e., tutorial.RData). The Linux term "directory" is equivalent to "folder" in Windows.

Upon login onto Cheaha, there is a table summarizing the partition categories on the HPC dashboard (similar to the partition table in Figure 5; not captured in Figure 2). You can also see such a partition table in the Cheaha shell terminal. Please note that your HPC may have a different set of partition names and different specifications depending on the capacity of your HPC. This is essential information when you will request compute node resources using the Simple Linux Utility for Resource Management (SLURM) job scheduler. In the case that your HPC does not have this summary table for partitions on the web portal and/or login node front page, you can always find out the detailed information about your HPC partitions using the following code,

scontrol show partition # This outputs information about all partitions of your HPC (Figure 4). You can print out information about a specific partition by specifying the name of the partition, for example, "scontrol show partition short" will output the information about the *short* partition of Cheaha.



```
[kejinhu@login004 ~]$ pwd # Print the path to the working directory
/home/kejinhu
[kejinhu@login004 ~]$ lscpu        # List CPU information for
Architecture:          x86_64       the current nodes
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                96
On-line CPU(s) list:   0-95
Thread(s) per core:    2
Core(s) per socket:    24
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 85
Model name:            Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz
Stepping:              7
CPU MHz:               3699.829
CPU max MHz:           4000.0000
CPU min MHz:           1200.0000
BogoMIPS:              6000.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              36608K
NUMA node0 CPU(s):     0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30
NUMA node1 CPU(s):     1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic se
 rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl x
e3 sdbg fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic movbe po
vpcid_single intel_ppin intel_pt ssbd mba ibrs ibpb stibp ibrs_en
vpcid rtm cqm mpx rdt_a avx512f avx512dq rdseed adx smap clflusho
qm_mbm_local dtherm ida arat pln pts pku ospke avx512_vnni md_cle
[kejinhu@login004 ~]$ ls # List files/directories in the working directory
Desktop  Documents  Downloads  Music  Pictures  Public  R  Templa
[kejinhu@login004 ~]$
```

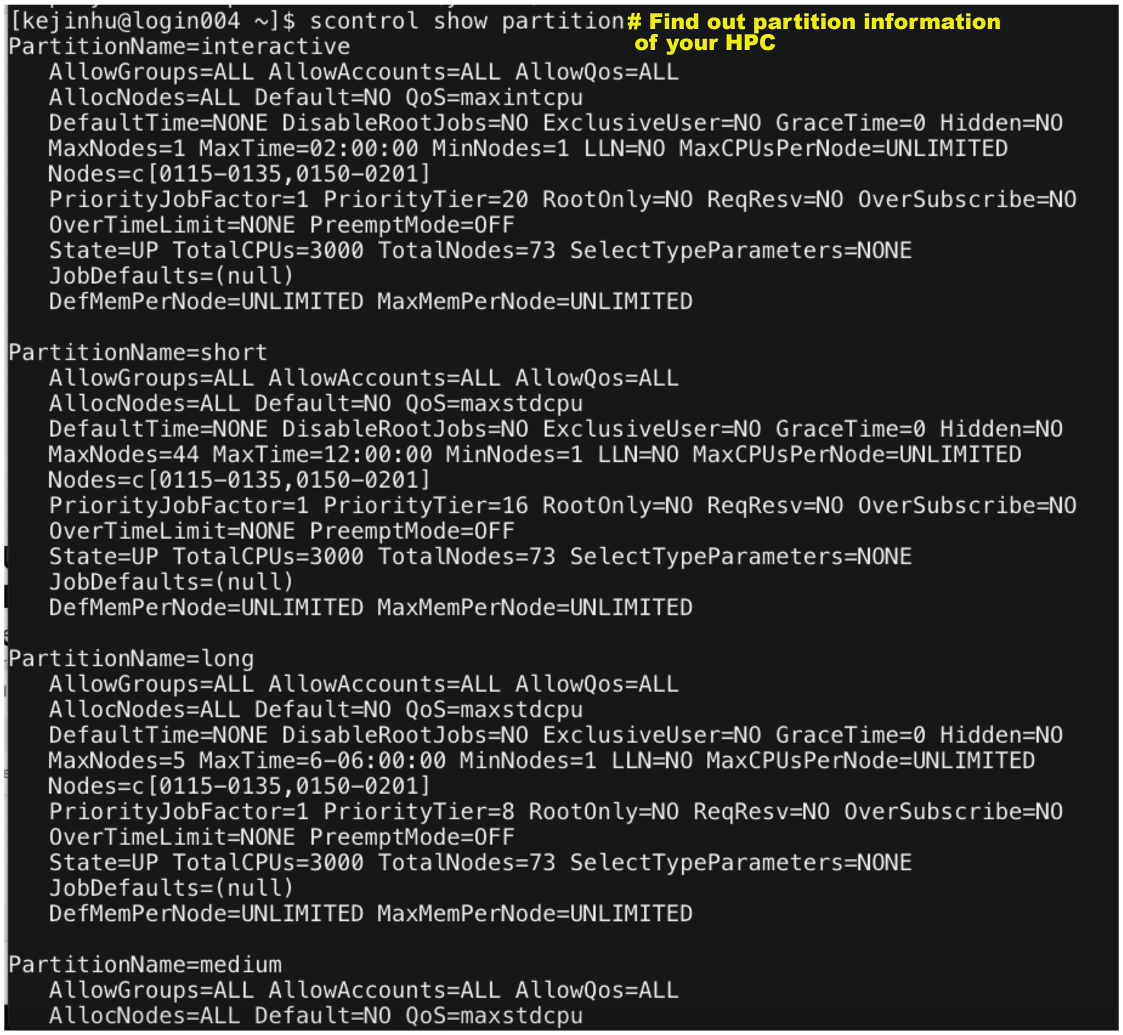
**Figure 3**. HPC login node terminal and HPC home directory. Login node name is underlined in red. The lscpu command reveals that the Cheaha login nodes have 2 nodes and 96 CPUs. The ~ after the login node name indicates that you are in your home directory. Each Linux command is described in yellow text immediately after the command as marked by a #.

Alternatively, you can also login your HPC account via the traditional secure shell (SSH) on your desktop device terminal (Figure 5). Upon login, the commands are the same regardless the device you use or the interfaces (web portal or SSH) although the screen appearances may vary.

> ssh kejinhu@cheaha.rc.uab.edu          # The syntax is ssh user_name@hostname.
> After this command, the HPC system will ask you for the password. Please do not be nervous when you do not see the password during typing because they do not show (top in Figure 5).

We will conduct the analyses and save the results in the data directory. You use the cd (Change Directory) command to change the working directory from your home directory to your data directory (Figure 6). For convenience, I use my data directory on Cheaha, */data/user/kejinhu* as an example; therefore, I issue cd /data/user/kejinhu in my HPC home directory,

> cd /data/user/kejinhu # This takes me to the */data/user/kejinhu* directory (the second command in Figure 6). For additional usage of the cd command, see the 6th and 9th commands. Please note that the home directory sign ~ has been replaced by the subdirectory names of each data subdirectory, i.e., *kejinhu*, *RNA_seq_tutorial*, and *fastq_files* in Figure 6. Upon in these directories, you can use the list command ls to see their contents (the 3rd, fifth, and 8th command in Figure 6).



```
[kejinhu@login004 ~]$ scontrol show partition# Find out partition information
                                                of your HPC
PartitionName=interactive
   AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
   AllocNodes=ALL Default=NO QoS=maxintcpu
   DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0 Hidden=NO
   MaxNodes=1 MaxTime=02:00:00 MinNodes=1 LLN=NO MaxCPUsPerNode=UNLIMITED
   Nodes=c[0115-0135,0150-0201]
   PriorityJobFactor=1 PriorityTier=20 RootOnly=NO ReqResv=NO OverSubscribe=NO
   OverTimeLimit=NONE PreemptMode=OFF
   State=UP TotalCPUs=3000 TotalNodes=73 SelectTypeParameters=NONE
   JobDefaults=(null)
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED

PartitionName=short
   AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
   AllocNodes=ALL Default=NO QoS=maxstdcpu
   DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0 Hidden=NO
   MaxNodes=44 MaxTime=12:00:00 MinNodes=1 LLN=NO MaxCPUsPerNode=UNLIMITED
   Nodes=c[0115-0135,0150-0201]
   PriorityJobFactor=1 PriorityTier=16 RootOnly=NO ReqResv=NO OverSubscribe=NO
   OverTimeLimit=NONE PreemptMode=OFF
   State=UP TotalCPUs=3000 TotalNodes=73 SelectTypeParameters=NONE
   JobDefaults=(null)
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED

PartitionName=long
   AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
   AllocNodes=ALL Default=NO QoS=maxstdcpu
   DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0 Hidden=NO
   MaxNodes=5 MaxTime=6-06:00:00 MinNodes=1 LLN=NO MaxCPUsPerNode=UNLIMITED
   Nodes=c[0115-0135,0150-0201]
   PriorityJobFactor=1 PriorityTier=8 RootOnly=NO ReqResv=NO OverSubscribe=NO
   OverTimeLimit=NONE PreemptMode=OFF
   State=UP TotalCPUs=3000 TotalNodes=73 SelectTypeParameters=NONE
   JobDefaults=(null)
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED

PartitionName=medium
   AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
   AllocNodes=ALL Default=NO QoS=maxstdcpu
```

**Figure 4**. Find out the partition information of your HPC using the scontrol show partition command. The screenshot captured information for some of the partitions only (*interactive*, *short*, *long*, and a little of *medium*).

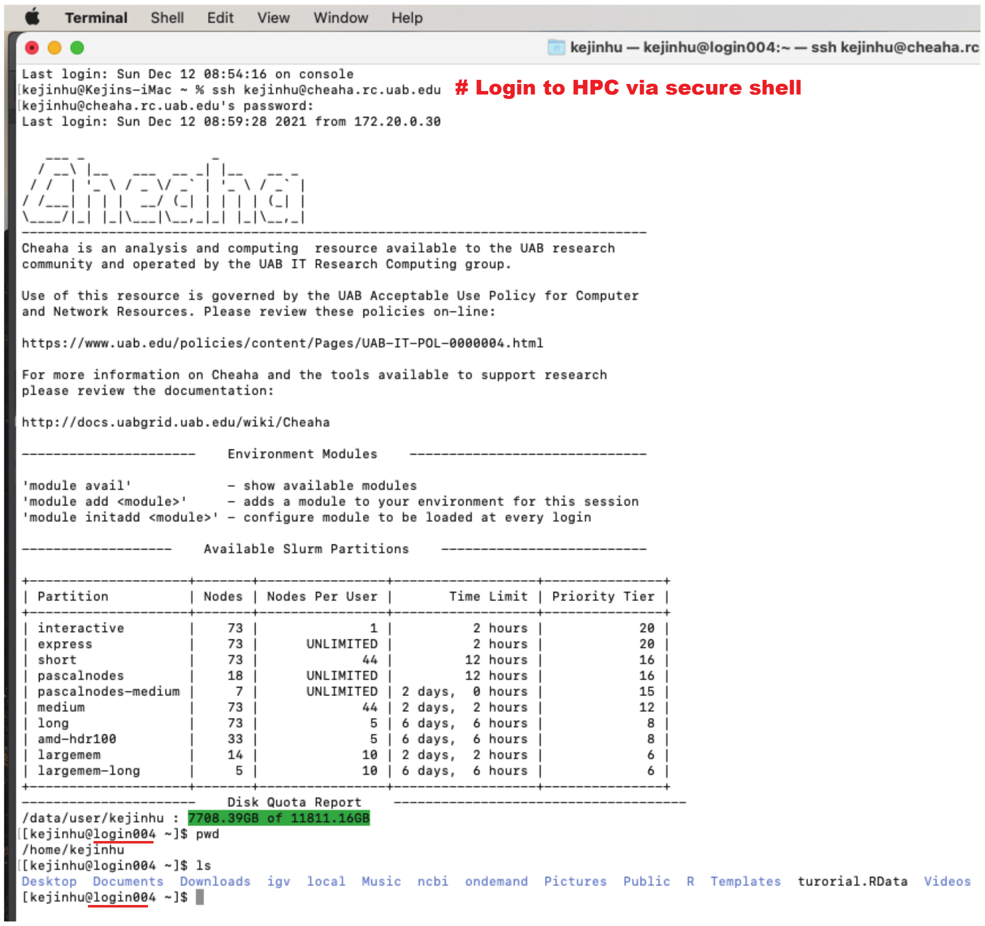
Under your data directory of HPC account, generate a directory of RNA_seq_tutorial using the "make directory" command mkdir (the fourth command in Figure 6), mkdir RNA_seq_tutorial # This generates a directory of RNA_seq_tutorial within which you will further make a subdirectory of fastq_files later.

cd RNA_seq_tutorial # This takes you into the RNA_seq_tutorial directory (the sixth command in Figure 6).

Then, prepare a subdirectory of fastq_files within the RNA_seq_tutorial directory (the 7th command in Figure 6),

mkdir fastq_files # This generates a fastq_files subdirectory into which you will transfer your original RNA-seq data files in FASTQ format (Figure 6).

cd fastq_files # This changes the working directory from the current working directory of RNA_seq_tutorial to the fastq_files directory (the 9th command in Figure 6).


Terminal Shell Edit View Window Help
kejinhu — kejinhu@login004:~ — ssh kejinhu@cheaha.rc

```
Last login: Sun Dec 12 08:54:16 on console
kejinhu@Kejins-iMac ~ % ssh kejinhu@cheaha.rc.uab.edu  # Login to HPC via secure shell
kejinhu@cheaha.rc.uab.edu's password:
Last login: Sun Dec 12 08:59:28 2021 from 172.20.0.30

   ___ _                _
  / __\ |__   ___  __ _| |__   __ _
 / /  | '_ \ / _ \/ _` | '_ \ / _` |
/ /___| | | |  __/ (_| | | | | (_| |
\____/|_| |_|\___|\__,_|_| |_|\__,_|
-------------------------------------------------------------------------------
Cheaha is an analysis and computing  resource available to the UAB research
community and operated by the UAB IT Research Computing group.

Use of this resource is governed by the UAB Acceptable Use Policy for Computer
and Network Resources. Please review these policies on-line:

https://www.uab.edu/policies/content/Pages/UAB-IT-POL-0000004.html

For more information on Cheaha and the tools available to support research
please review the documentation:

http://docs.uabgrid.uab.edu/wiki/Cheaha

----------------------   Environment Modules   ------------------------------

'module avail'           - show available modules
'module add <module>'    - adds a module to your environment for this session
'module initadd <module>' - configure module to be loaded at every login

-------------------   Available Slurm Partitions   --------------------------

+---------------------+-------+----------------+-------------------+---------------+
| Partition           | Nodes | Nodes Per User |        Time Limit | Priority Tier |
+---------------------+-------+----------------+-------------------+---------------+
| interactive         |    73 |              1 |           2 hours |            20 |
| express             |    73 |      UNLIMITED |           2 hours |            20 |
| short               |    73 |             44 |          12 hours |            16 |
| pascalnodes         |    18 |      UNLIMITED |          12 hours |            16 |
| pascalnodes-medium  |     7 |      UNLIMITED | 2 days,   0 hours |            15 |
| medium              |    73 |             44 | 2 days,   2 hours |            12 |
| long                |    73 |              5 | 6 days,   6 hours |             8 |
| amd-hdr100          |    33 |              5 | 6 days,   6 hours |             8 |
| largemem            |    14 |             10 | 2 days,   2 hours |             6 |
| largemem-long       |     5 |             10 | 6 days,   6 hours |             6 |
+---------------------+-------+----------------+-------------------+---------------+
----------------------   Disk Quota Report   -------------------------------------
/data/user/kejinhu : 7708.39GB of 11811.16GB
[kejinhu@login004 ~]$ pwd
/home/kejinhu
[kejinhu@login004 ~]$ ls
Desktop  Documents  Downloads  igv  local  Music  ncbi  ondemand  Pictures  Public  R  Templates  turorial.RData  Videos
[kejinhu@login004 ~]$
```

**Figure 5.** Login HPC via SSH on an iMAC terminal. The partition table is also captured in this screenshot. You can see the same table but with slightly different appearance when you login via the web interface. ~ indicates the home directory.

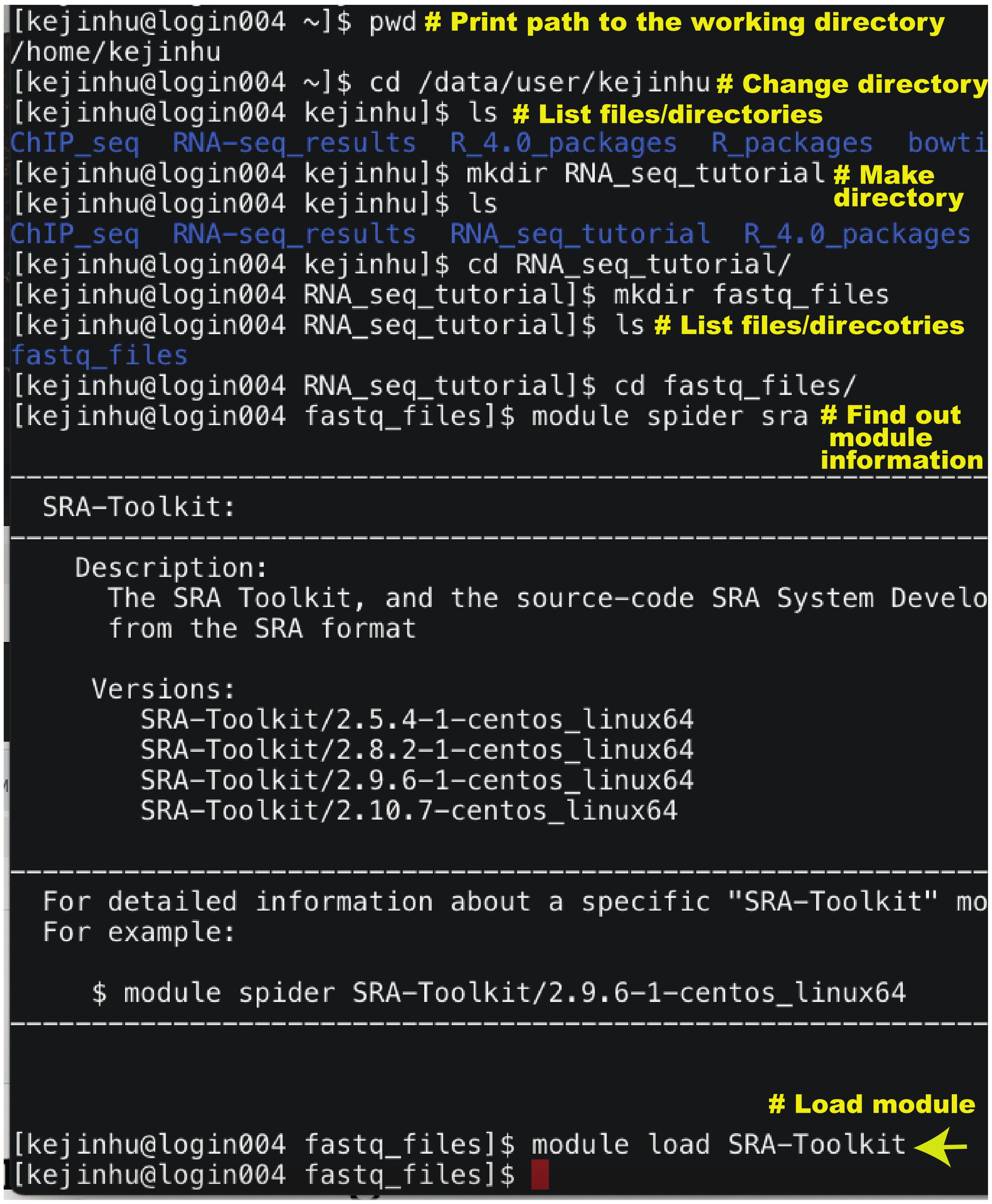
## 2.2 Find and load a module on HPC with the SRA-Toolkit module as an example

To analyze RNA-seq data on HPC, we first need to move the original raw data in FASTQ files onto your HPC. To this end, a software or a module is required. Your FASTQ files may be stored in various places. The best tools for transferring data to HPC depend on where the FASTQ files are stored. In this tutorial, we will download the RNA-seq data from the Sequence Read Archive (SRA) repository. The customized tool for downloading data from the SRA repository is the SRA-Toolkit. Your HPC should have the SRA-Toolkit module installed already if your institution has a community of bioinformaticians. Many versions of a module are usually installed on any HPC, and the command module spider is very helpful to find out the available versions of a specific software.

> module spider sra # This returns a list of SRA-Toolkit of various versions installed on your HPC (in the lower middle of Figure 6). The command module avail functions similarly (see below).

Then we load the SRA-Toolkit using the command of module load (Figures 6 and 8). "module load" is the command equivalent to "clicking on an application icon" in Windows. In Windows system, we click to invoke an application software (for example, Excel or Word) and their specific functions. In Linux, we use command lines to communicate with the computer.

> module load SRA-Toolkit # This loads the default version of the SRA-Toolkit (Figures 6 and 8). Please note that the module name for the module load command is case sensitive, but it is not for the command of module spider or module avail (see below). If you encounter issues in loading the SRA-Toolkit module (or other software), please try a different spelling. The best practice is that you copy the entire module name with the version information and paste them after the module load command, for example, module load SRA-Toolkit/2.10.7-centos_linux64. You can call module avail sra after loading it, and you will see which version has been loaded. The loaded version is indicated by an "L". When you do not include the version number the default version is loaded. The default version is indicated by a "D" when you output the module information using "module avail". For module spider, you do not even need to spell out the full name of a software package, for example, you can issue module spider sr, and it will return all the SRA-Toolkit versions available on your HPC. This is also true for module avail. Try module avail sr.



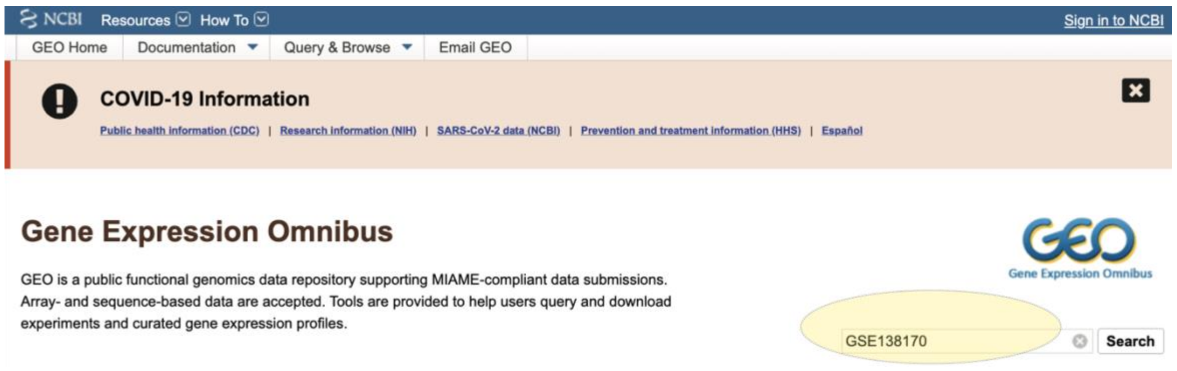
```
[kejinhu@login004 ~]$ pwd # Print path to the working directory
/home/kejinhu
[kejinhu@login004 ~]$ cd /data/user/kejinhu # Change directory
[kejinhu@login004 kejinhu]$ ls # List files/directories
ChIP_seq  RNA-seq_results  R_4.0_packages  R_packages  bowti
[kejinhu@login004 kejinhu]$ mkdir RNA_seq_tutorial # Make
[kejinhu@login004 kejinhu]$ ls                       directory
ChIP_seq  RNA-seq_results  RNA_seq_tutorial  R_4.0_packages
[kejinhu@login004 kejinhu]$ cd RNA_seq_tutorial/
[kejinhu@login004 RNA_seq_tutorial]$ mkdir fastq_files
[kejinhu@login004 RNA_seq_tutorial]$ ls # List files/direcotries
fastq_files
[kejinhu@login004 RNA_seq_tutorial]$ cd fastq_files/
[kejinhu@login004 fastq_files]$ module spider sra # Find out
                                                     module
                                                     information
------------------------------------------------------------
  SRA-Toolkit:
------------------------------------------------------------
    Description:
      The SRA Toolkit, and the source-code SRA System Develo
      from the SRA format

     Versions:
        SRA-Toolkit/2.5.4-1-centos_linux64
        SRA-Toolkit/2.8.2-1-centos_linux64
        SRA-Toolkit/2.9.6-1-centos_linux64
        SRA-Toolkit/2.10.7-centos_linux64

------------------------------------------------------------
  For detailed information about a specific "SRA-Toolkit" mo
  For example:

     $ module spider SRA-Toolkit/2.9.6-1-centos_linux64
------------------------------------------------------------


                                                 # Load module
[kejinhu@login004 fastq_files]$ module load SRA-Toolkit
[kejinhu@login004 fastq_files]$
```

**Figure 6.** Find out and load Linux modules using the commands of module spider and module load. Other Linux commands were also captured in this screenshot. Please note that this screenshot is not a full-screen screenshot, and some information was cut off. The yellow texts are illustrations by the author.

## 2.3 Locate the SRA data files in the repository to download

In this tutorial, we will use the RNA-seq data set of human embryonic stem cells (hESCs) and human fibroblast BJ cells with the Gene Expression Omnibus (GEO) series access code of GSE138170 [11]. In the GEO accession query box at https://www.ncbi.nlm.nih.gov/geo/ , you search for GSE138170 (shaded in yellow in Figure 7). When you open the GEO page for this dataset, you will see there are 7 samples under the Samples entry (on the left column of the page). Click the HTML text More (or the plus sign + thumb nail by the More; they will become "- Less" after clicking), and you will see the GEO sample (GSM) codes for the 7 samples (GSM4101203 – GSM4101209). After you click any of these 7 HTML sample codes, you will see a page containing information about the selected RNAs-seq data. At the bottom of this page, you can see the SRX codes. The SRX codes for these 7 RNA-seq are: SRX6923623 - SRX6923629. Click one of the HTML SRX codes and you will see a new page about the selected sample. At the bottom of the page, you will see the SRR code in a table. The 7 SRR accession codes are: SRR10203569 - SRR10203575. We will use these SRR codes to download the data sets using methods introduced below.

Alternatively, under the "Relations" section on the GEO series code (GSE) page, click on the clickable HTML SRA project code (SRP code) and you will land on a page with the list of the GEO sample codes. Now, you can also locate each SRA code via the GEO sample codes described above.



GEO is a public functional genomics data repository supporting MIAME-compliant data submissions. Array- and sequence-based data are accepted. Tools are provided to help users query and download experiments and curated gene expression profiles.

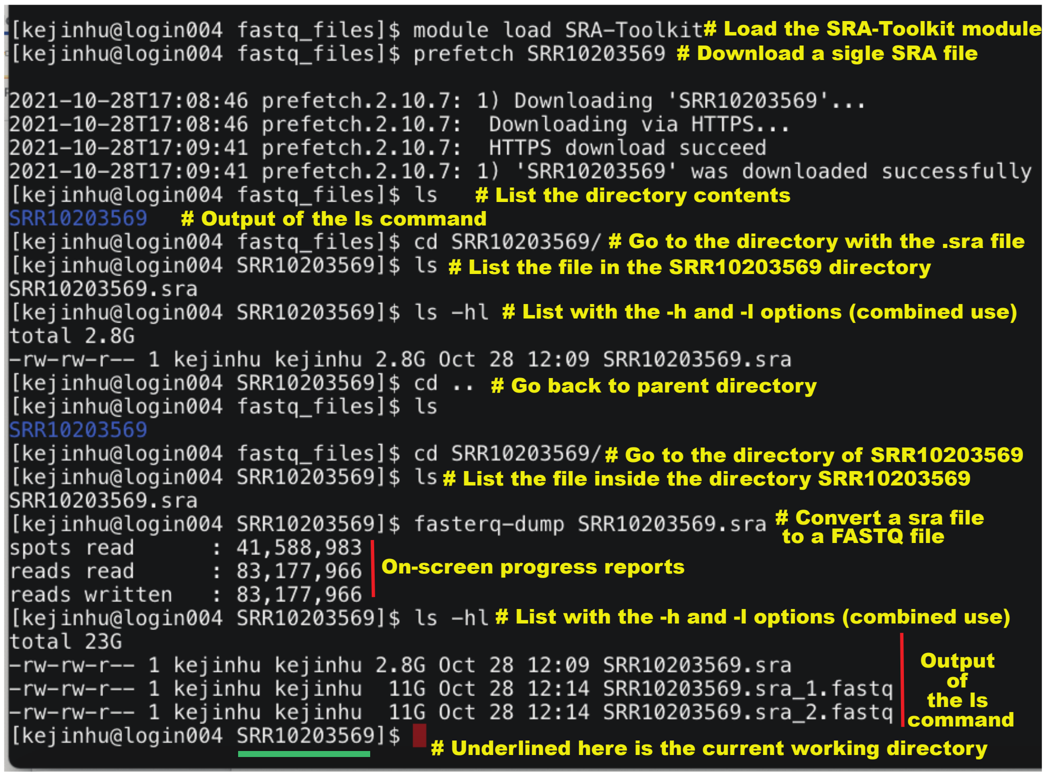
**Figure 7.** Find out the RNA-seq data in the GEO repository to download. The GSE138170 accession code in the GEO search box is shaded in yellow.

## 2.4 The prefetch command for downloading the SRA files

For the purpose of demonstration, this section introduces downloading each data file using the SRA-Toolkit command prefetch on both the login nodes (not encouraged) and a compute node (see below),

```
prefetch SRR10203569          # This downloads the data for the SRR10203569
sample (Figure 8). It automatically generates a directory with the entry code as the
```

directory name containing the file of SRR10203569.sra. Use the command ls to see the resulting directory of SRR10203569. You can go inside the SRR10203569 directory using cd SRR10203569, and see the file using the ls command (Figure 8). To come back to the parent directory, i.e., *fast_files* directory, simply issue "cd ..". The".." here represents parent directory (Figure 8). You can download several files at one command, for example, prefetch SRR10203569 SRR10203570. This single code would download the files for these two samples.  You can find the help page of the prefetch command by calling, prefetch --help after you load the SRA-Toolkit module.



```
[kejinhu@login004 fastq_files]$ module load SRA-Toolkit# Load the SRA-Toolkit module
[kejinhu@login004 fastq_files]$ prefetch SRR10203569 # Download a sigle SRA file

2021-10-28T17:08:46 prefetch.2.10.7: 1) Downloading 'SRR10203569'...
2021-10-28T17:08:46 prefetch.2.10.7:  Downloading via HTTPS...
2021-10-28T17:09:41 prefetch.2.10.7:  HTTPS download succeed
2021-10-28T17:09:41 prefetch.2.10.7: 1) 'SRR10203569' was downloaded successfully
[kejinhu@login004 fastq_files]$ ls    # List the directory contents
SRR10203569    # Output of the ls command
[kejinhu@login004 fastq_files]$ cd SRR10203569/ # Go to the directory with the .sra file
[kejinhu@login004 SRR10203569]$ ls # List the file in the SRR10203569 directory
SRR10203569.sra
[kejinhu@login004 SRR10203569]$ ls -hl # List with the -h and -l options (combined use)
total 2.8G
-rw-rw-r-- 1 kejinhu kejinhu 2.8G Oct 28 12:09 SRR10203569.sra
[kejinhu@login004 SRR10203569]$ cd .. # Go back to parent directory
[kejinhu@login004 fastq_files]$ ls
SRR10203569
[kejinhu@login004 fastq_files]$ cd SRR10203569/# Go to the directory of SRR10203569
[kejinhu@login004 SRR10203569]$ ls# List the file inside the directory SRR10203569
SRR10203569.sra
[kejinhu@login004 SRR10203569]$ fasterq-dump SRR10203569.sra # Convert a sra file
                                                               to a FASTQ file
spots read       : 41,588,983
reads read       : 83,177,966  On-screen progress reports
reads written    : 83,177,966
[kejinhu@login004 SRR10203569]$ ls -hl # List with the -h and -l options (combined use)
total 23G
-rw-rw-r-- 1 kejinhu kejinhu 2.8G Oct 28 12:09 SRR10203569.sra          Output
-rw-rw-r-- 1 kejinhu kejinhu  11G Oct 28 12:14 SRR10203569.sra_1.fastq    of
-rw-rw-r-- 1 kejinhu kejinhu  11G Oct 28 12:14 SRR10203569.sra_2.fastq  the ls
[kejinhu@login004 SRR10203569]$                                          command
                                  # Underlined here is the current working directory
```

**Figure 8.** Download RNA-seq data from SRA repository using the prefetch command, and convert the SRA files into FASTQ files using the fasterq-dump command. The subdirectory generated automatically is in blue text. The yellow texts in the screenshot are illustrations added by the author.

## 2.5 Use the fasterq-dump command to convert SRA files into FASTQ ones

After you run the above code on the login nodes, you will see a subdirectory with a name of the sample code under the directory of fastq_files. You can convert the SRA files into FASTQ files using the SRA-Toolkit command fasterq-dump on the login node (not encouraged) outside the subdirectory,

fasterq-dump SRR10203569 # This generates two FASTQ files (SRR10203569_1.fastq and SRR10203569_2.fastq) outside the SRR10203569 subdirectory under the fastq_files directory because the RNA-seq protocol is paired-end sequencing (bottom in Figure 8). Please note that SRR10203569 here is the directory name instead of the file name when you will convert inside the subdirectory (see below). Call fasterq-dump --help to find more information about its use.

You can also make the conversion within the SRR10203569 subdirectory. Go to the subdirectory, and you will see an SRA file within it. You can convert the SRA file into FASTQ files using the command of fasterq-dump (Figure 8),

cd SRR10203569 # This takes you to the subdirectory of SRR10203569.

fasterq-dump SRR10203569.sra # This also converts the SRA file into two FASTQ files, i.e., SRR10203569_1.fastq and SRR10203569_2.fastq (Figure 8). But, these two FASTQ files are located within the SRR10203569 subdirectory. The code is the same on the login node and the compute node (see below). Please note that you use the file name SRR10203569.sra here (instead of the subdirectory name when you convert outside the subdirectory (see above).

You can see the converted FASTQ files using the list command ls with the -h and -l options (the last command in Figure 8),

ls -hl # The -l option means "long format" and the -h option means the file sizes are displayed in the human readable form. The -l and -h options can be used separately (-h -l) or together as -hl.

Before we move to the next step, we remove the downloaded files and subdirectory using the generic rm (remove) command so that we can experience the downloading of the same item again on the compute nodes.

cd .. # This will bring you back to the parent directory, i.e., the fastq_files directory. In Linux, double dots (..) mean parent directory of the current directory, and a single dot denotes the current directory.

rm SRR10203569_1.fastq SRR10203569_2.fastq # This removes the two files listed after rm and rm can remove all files listed after it.

Alternatively, we can use a wild card * here to remove the two files altogether with the same extension of .fastq using a simpler code,

rm *.fastq # This removes all files with the extension of ".fastq"within the current working directory. Make sure this is what you want to do.

You still have another copy of the FASTQ files and the subdirectory of SRR10203569 since we have converted twice. Now remove the subdirectory and the two FASTQ files within it using the rm command,

```
rm -r SRR10203569     # This removes the directory of SRR10203569 and all the files within it. Unlike deleting a file(s), you need to use the recursive -r option when you remove a directory even though it is empty.
```

Use the list command ls to confirm that the subdirectory and files are deleted,

```
ls
```

## 2.6 Work and download FASTQ files on a pseudo terminal

### 2.6.1 Request compute node resources

It is not encouraged (literally not allowed) to download large files on the login nodes of an HPC because there are limited resources on the login nodes that are dedicated for HPC login, light file managements, code editing, and job submission only [12]. We should download the FASTQ files or conduct any other intensive computation on the compute nodes. To this end, we first need to request compute node resources because you are on the login nodes when you log in to your HPC.

There are many different resource managers. Here we use the free open-source and most dominant resource management system SLURM (Simple Linux Utility for Resource Management) [13, 14]. Since this downloading is not a very heavy job, we can try to request an interactive job using the srun command. If your HPC uses other software for resource management (e.g., Portable Batch System, OpenLava, LoadLeveler, and others), you may need training from your HPC team or study the related tutorials. Upon allocation of the compute resources (nodes, CPUs, time, memory, and others), the application scripts discussed here are the same regardless the resource management systems used.

```
srun --cpus-per-task=2 --mem=10G --partition=express --pty /bin/bash
```

The above code will generate a pseudo terminal interface on which you can work on compute nodes with the allocated resources (Figure 9). On the compute node, you can work like on the login node, but you do not consume the resources specified for login to HPC by all users. In the above code, we request 2 CPUs (using the one-letter option of -c or the equivalent word option of --cpus-per-task), 10 G of minimum memory (using the option of --mem), and the *express* partition. You can also use the --mem-per-cpu option to request memory per CPU. The two options --mem and --mem-per-cpu are related but different. In the latter case your total minimum memory (defined by the --mem option) is the product of memory/CPU and the number of CPUs. Please note that the partition names on your HPC are likely different from that of Cheaha. The --pty option means initiating a pseudo terminal. The /bin/bash is a srun command, which means your codes will be interpreted by bash. You may just call, srun --pty bash, and this will bring you to the psedoterminal bash shell with the default amount of compute resources. After you run the above code, you will see you are not on your login node anymore and the prompt will change from the login node (kejinhu@login004 in my case) to a compute node (kejinhu@cNNNN in my case) (Figure 9).

To exit the pseudo terminal, you just call the command exit. When you run the exit command you give up your allocated compute resources even though it is before your requested time limit. Or you can use the scancel SLURM command. To do this, you need to find out the job ID first using, echo $SLURM_JOB_ID, and then scancel NNNNNNNN (the job ID you identified by the echo command). The job ID can also be found using, squeue -u kejinhu (your username of HPC; the -u option means user). Please note that if you just click the tab on the web interface to close the pseudo terminal, you are still holding the srun resources. You need to use the exit command to give up the compute nodes you have requested using srun. The exit command just gives up the compute nodes, and you go back to the login nodes. You need to call exit again to leave the login terminal if you want.

To see the basic information about a SLURM job including the number of CPUs, and memory allocated, you can use the syntax below,

```
scontrol show job [job ID#]
```

When you simply request compute node resources for interactive tasks using srun --pty /bin/bash, or even srun --pty bash, Cheaha allocates one CPU from one node, and 1G of memory, at the *express* partition with a maximum wall time of 2 hours, by default. This resource is sufficient to download the 7 RNA-seq samples using the fasterq-dump command. My experience is that you should allow 1 minute to download 1 G of RNA-seq data using the fasterq-dump command with the srun-requested resources on Cheaha by default. The command fasterq-dump is faster and simpler to use than the old fastq-dump. The --time option can be used with srun to require more time (> 2 hours).

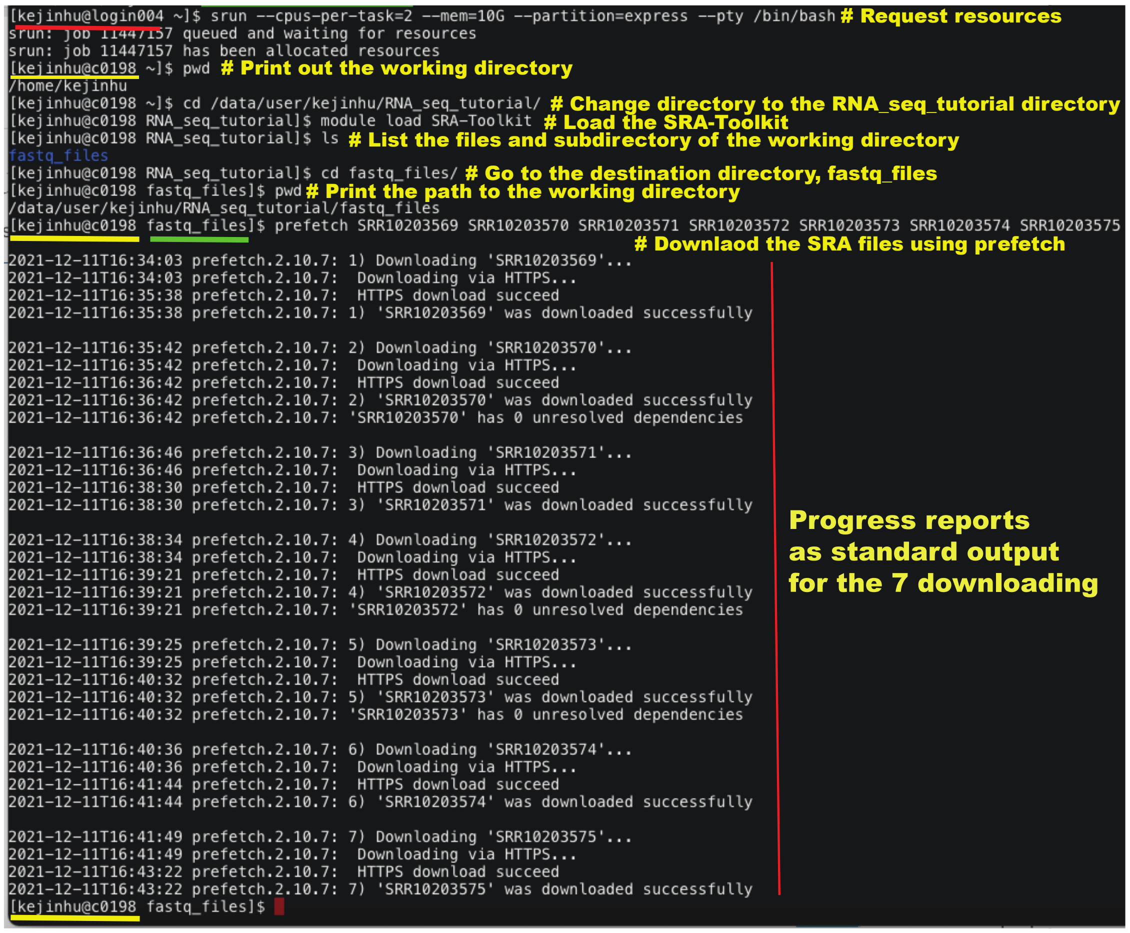
Once compute nodes are allocated to you, you can find out the information about the nodes using the lscpu command. The command nproc --all will print out the number of processors on the allocated nodes. You can find out the node names that assigned to you using the hostname command. Of course, you can also use these three commands to find the information about the login nodes. But you use scontrol show job [job ID#] to find out the resource information allocated to you as introduced above.

### 2.6.2 Download and convert SRA files in two steps (faster procedure)

With the requested resources using the SLURM command srun, we can download the files on the compute node in interactive mode (Figures 9 and 10),

module spider sra

module load SRA-Toolkit     # If you have loaded the package on the login nodes the information is passed to the compute nodes you have just requested using srun, and you do not need to re-load the package again on the compute nodes.



```
[kejinhu@login004 ~]$ srun --cpus-per-task=2 --mem=10G --partition=express --pty /bin/bash # Request resources
srun: job 11447157 queued and waiting for resources
srun: job 11447157 has been allocated resources
[kejinhu@c0198 ~]$ pwd # Print out the working directory
/home/kejinhu
[kejinhu@c0198 ~]$ cd /data/user/kejinhu/RNA_seq_tutorial/ # Change directory to the RNA_seq_tutorial directory
[kejinhu@c0198 RNA_seq_tutorial]$ module load SRA-Toolkit # Load the SRA-Toolkit
[kejinhu@c0198 RNA_seq_tutorial]$ ls # List the files and subdirectory of the working directory
fastq_files
[kejinhu@c0198 RNA_seq_tutorial]$ cd fastq_files/ # Go to the destination directory, fastq_files
[kejinhu@c0198 fastq_files]$ pwd # Print the path to the working directory
/data/user/kejinhu/RNA_seq_tutorial/fastq_files
[kejinhu@c0198 fastq_files]$ prefetch SRR10203569 SRR10203570 SRR10203571 SRR10203572 SRR10203573 SRR10203574 SRR10203575
                                                                    # Downlaod the SRA files using prefetch
2021-12-11T16:34:03 prefetch.2.10.7: 1) Downloading 'SRR10203569'...
2021-12-11T16:34:03 prefetch.2.10.7:  Downloading via HTTPS...
2021-12-11T16:35:38 prefetch.2.10.7:  HTTPS download succeed
2021-12-11T16:35:38 prefetch.2.10.7: 1) 'SRR10203569' was downloaded successfully

2021-12-11T16:35:42 prefetch.2.10.7: 2) Downloading 'SRR10203570'...
2021-12-11T16:35:42 prefetch.2.10.7:  Downloading via HTTPS...
2021-12-11T16:36:42 prefetch.2.10.7:  HTTPS download succeed
2021-12-11T16:36:42 prefetch.2.10.7: 2) 'SRR10203570' was downloaded successfully
2021-12-11T16:36:42 prefetch.2.10.7: 'SRR10203570' has 0 unresolved dependencies

2021-12-11T16:36:46 prefetch.2.10.7: 3) Downloading 'SRR10203571'...
2021-12-11T16:36:46 prefetch.2.10.7:  Downloading via HTTPS...
2021-12-11T16:38:30 prefetch.2.10.7:  HTTPS download succeed
2021-12-11T16:38:30 prefetch.2.10.7: 3) 'SRR10203571' was downloaded successfully

2021-12-11T16:38:34 prefetch.2.10.7: 4) Downloading 'SRR10203572'...
2021-12-11T16:38:34 prefetch.2.10.7:  Downloading via HTTPS...
2021-12-11T16:39:21 prefetch.2.10.7:  HTTPS download succeed
2021-12-11T16:39:21 prefetch.2.10.7: 4) 'SRR10203572' was downloaded successfully
2021-12-11T16:39:21 prefetch.2.10.7: 'SRR10203572' has 0 unresolved dependencies

2021-12-11T16:39:25 prefetch.2.10.7: 5) Downloading 'SRR10203573'...
2021-12-11T16:39:25 prefetch.2.10.7:  Downloading via HTTPS...
2021-12-11T16:40:32 prefetch.2.10.7:  HTTPS download succeed
2021-12-11T16:40:32 prefetch.2.10.7: 5) 'SRR10203573' was downloaded successfully
2021-12-11T16:40:32 prefetch.2.10.7: 'SRR10203573' has 0 unresolved dependencies

2021-12-11T16:40:36 prefetch.2.10.7: 6) Downloading 'SRR10203574'...
2021-12-11T16:40:36 prefetch.2.10.7:  Downloading via HTTPS...
2021-12-11T16:41:44 prefetch.2.10.7:  HTTPS download succeed
2021-12-11T16:41:44 prefetch.2.10.7: 6) 'SRR10203574' was downloaded successfully

2021-12-11T16:41:49 prefetch.2.10.7: 7) Downloading 'SRR10203575'...
2021-12-11T16:41:49 prefetch.2.10.7:  Downloading via HTTPS...
2021-12-11T16:43:22 prefetch.2.10.7:  HTTPS download succeed
2021-12-11T16:43:22 prefetch.2.10.7: 7) 'SRR10203575' was downloaded successfully
[kejinhu@c0198 fastq_files]$
```

**Figure 9**. Download SRA files using prefetch on the compute nodes in interactive mode. Please note that the operation is on the c0198 compute node (kejinhu@c0198, yellow underline), not on the login node (kejinhu@login004, red underline). Yellow texts marked with a # are description of the codes. We run prefetch in the directory of fastq_files (green underline) because it is in the interactive mode. In the interactive mode, you can see the downloading progress report on your screen (red vertical line).

cd /data/user/kejinhu/RNA_seq_tutorial/fastq_files/     # Please note that you are in the directory where you run the srun command. If this directory is not what you want to work in, you should change your directory to the one you want using the cd command.

prefetch SRR10203569 SRR10203570 SRR10203571 SRR10203572 SRR10203573 SRR10203574 SRR10203575 # It took only 4 minutes to download the 7 SRA files using the prefetch command with the default resources on Cheaha (1 CPU, 1G of memory). However, the time will vary depending on the time of downloading. On weekend, it may be faster and in the peak times it may take longer.

fasterq-dump SRR10203569 SRR10203570 SRR10203571 SRR10203572 SRR10203573 SRR10203574 SRR10203575 # To convert all the 7 SRA files here into FASTQ files, it took 15 minutes only using the Cheaha default resources (1 CPU, 1G memory on Cheaha). Therefore, using the two-step procedure the total wall time is 17 minutes vs 36 minutes using the one-step procedure (see below). The wall time is cut by half at least. At this step you can simply convert all SRA files into FASTQ files using a short code with the wild card, fasterq-dump SRR* since you have downloaded the SRR files onto your HPC already (Figure 10). But in the one-step download using the command fasterq-dump (below), you cannot use any wildcard with the access codes. Also, you cannot use the wildcard for the command prefetch at the previous step because SRA repository uses SRR for all entries.

### 2.6.3 Convert the downloaded SRA files using multiple threading

The SRA-Toolkit command fasterq-dump supports multiple threading. This means it can use multiple CPUs to conduct the calculation of divided portions of the job simultaneously. However, when we request compute node resources using the default setting for resources, srun --pty /bin/bash we are given one CPU only. But the default threading for the command fasterq-dump is 6 (i.e., --thread=6, or -e 6). Therefore, we can speed up the conversion by requesting more CPUs from the clusters. Let us try,

```
srun –cpus-per-task=6 --pty /bin/bash
```

Then, we use the same code as follows,

```
fasterq-dump SRR10203569 SRR10203570 SRR10203571 SRR10203572
SRR10203573 SRR10203574 SRR10203575
```

With 6 CPUs (the default value for the -e option) and the same fasterq-dump code, now it takes around 4 minutes only to convert all the 7 files (vs 15 minutes with one allocated CPU).

We can even shorten the time by requesting more CPUs than the default number of threads for the fasterq-dump command (--thread=6) and increasing the threading at the same time,

> srun --cpus-per-task=12 --pty /bin/bash # This script requests 12 CPUs with other resources as default.
>
> fasterq-dump -e 12 SRR10203569 SRR10203570 SRR10203571 SRR10203572 SRR10203573 SRR10203574 SRR10203575 # This code uses 12 CPUs for the conversion job. The code can be shortened as fasterq-dump -e SRR*

By requesting 12 CPUs (--cpus-per-task=12) and defining 12 threading (-e 12, or --thread=12) in the above codes, now it takes 2 minutes only to convert the 7 samples.

At this point, you have 7 subdirectories, and 14 FASTQ files which are located outside their original subdirectories inside the parent fastq_files directory. you can remove the 7 sub-directories since we will no longer need those files in the following analysis steps,

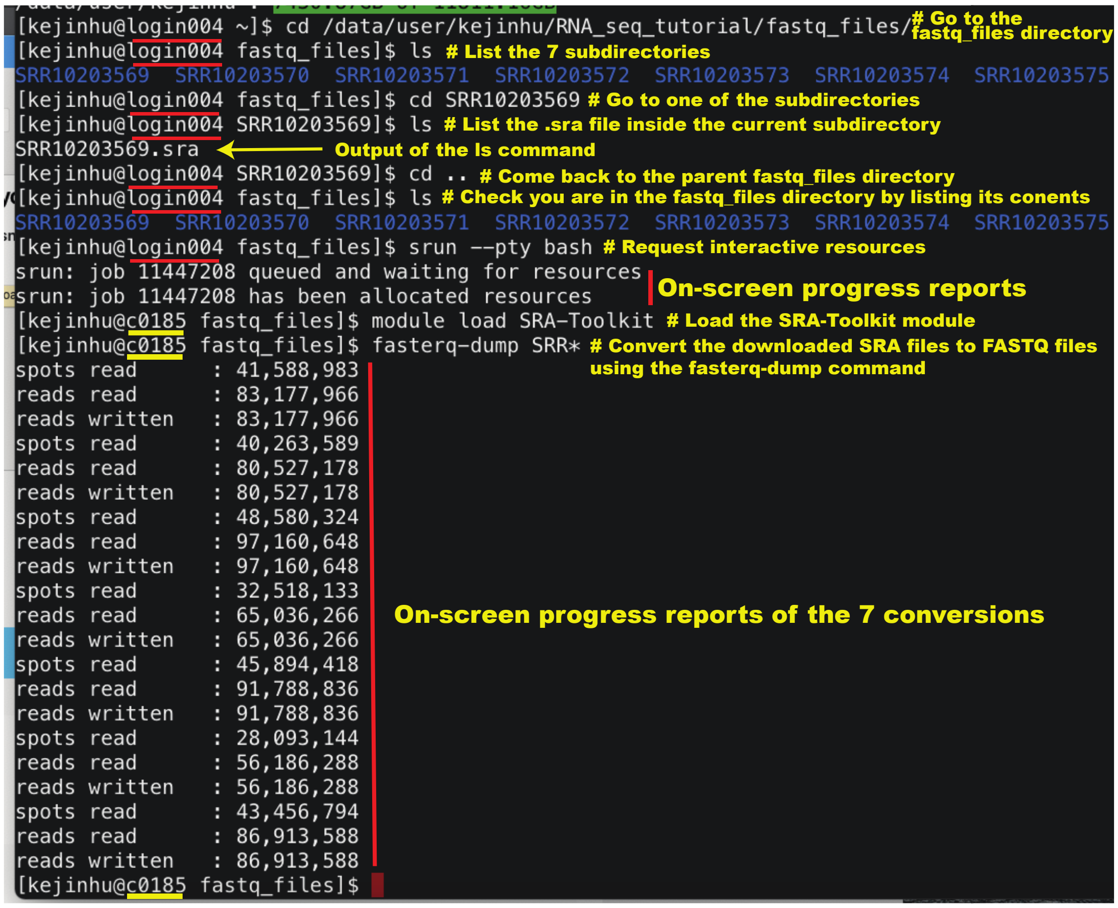
> rm -r SRR10203569 SRR10203570 SRR10203571 SRR10203572 SRR10203573 SRR10203574 SRR10203575 # Please note that you need to use the -r option to remove the directories and their contents, i.e. the SRA files. You can simply remove all the 7 subdirectories using a wildcard, that is, rm -r SRR*. Please note that this short code will also remove all the converted FASTQ files whose file names start with SRR as well. A wildcard denotes any character in a bash string.

### 2.6.4 download and convert SRA files in one step (slower procedure)

In fact, you can download and convert the SRA files at one step using the SRA-Toolkit fasterq-dump command, but many persons prefer the procedures of two steps described above because experience indicates that it is slower and unstable. Before practicing one-step downloading, let us first remove the FASTQ files that have been downloaded and converted in two steps introduced above if we did not use the short code rm -r SRR* to delete files/directories in the last step introduced,

> rm *.fastq # Using a wildcard like this, we can delete all the files with the same file extension of ".fastq"regardless the length of a file name. Use the ls command to confirm that the FASTQ files are deleted. If you have used the code of rm -r SRR* to delete the subdirectories you do not need this step because the FASTQ files shre the same SRR element, which is covered by SRR*.
>
> fasterq-dump SRR10203569 SRR10203570 SRR10203571 SRR10203572 SRR10203573 SRR10203574 SRR10203575 # This code downloads and converts the SRA files into the FASTQ files at one step even though there are no

pre-downloaded SRA files on your local HPC using the prefetch command. When you do not download the SRA files using prefetch beforehand you cannot use the wildcard * to download the SRR files. You need to specify all the entry codes in one-step downloading using the fasterq-dump command. Now, use the ls command to confirm that the files have been downloaded and converted.



```
[kejinhu@login004 ~]$ cd /data/user/kejinhu/RNA_seq_tutorial/fastq_files/# Go to the fastq_files directory
[kejinhu@login004 fastq_files]$ ls # List the 7 subdirectories
SRR10203569  SRR10203570  SRR10203571  SRR10203572  SRR10203573  SRR10203574  SRR10203575
[kejinhu@login004 fastq_files]$ cd SRR10203569 # Go to one of the subdirectories
[kejinhu@login004 SRR10203569]$ ls # List the .sra file inside the current subdirectory
SRR10203569.sra  Output of the ls command
[kejinhu@login004 SRR10203569]$ cd .. # Come back to the parent fastq_files directory
[kejinhu@login004 fastq_files]$ ls # Check you are in the fastq_files directory by listing its conents
SRR10203569  SRR10203570  SRR10203571  SRR10203572  SRR10203573  SRR10203574  SRR10203575
[kejinhu@login004 fastq_files]$ srun --pty bash # Request interactive resources
srun: job 11447208 queued and waiting for resources
srun: job 11447208 has been allocated resources      On-screen progress reports
[kejinhu@c0185 fastq_files]$ module load SRA-Toolkit # Load the SRA-Toolkit module
[kejinhu@c0185 fastq_files]$ fasterq-dump SRR* # Convert the downloaded SRA files to FASTQ files
                                                 using the fasterq-dump command
spots read      : 41,588,983
reads read      : 83,177,966
reads written   : 83,177,966
spots read      : 40,263,589
reads read      : 80,527,178
reads written   : 80,527,178
spots read      : 48,580,324
reads read      : 97,160,648
reads written   : 97,160,648
spots read      : 32,518,133
reads read      : 65,036,266   On-screen progress reports of the 7 conversions
reads written   : 65,036,266
spots read      : 45,894,418
reads read      : 91,788,836
reads written   : 91,788,836
spots read      : 28,093,144
reads read      : 56,186,288
reads written   : 56,186,288
spots read      : 43,456,794
reads read      : 86,913,588
reads written   : 86,913,588
[kejinhu@c0185 fastq_files]$
```

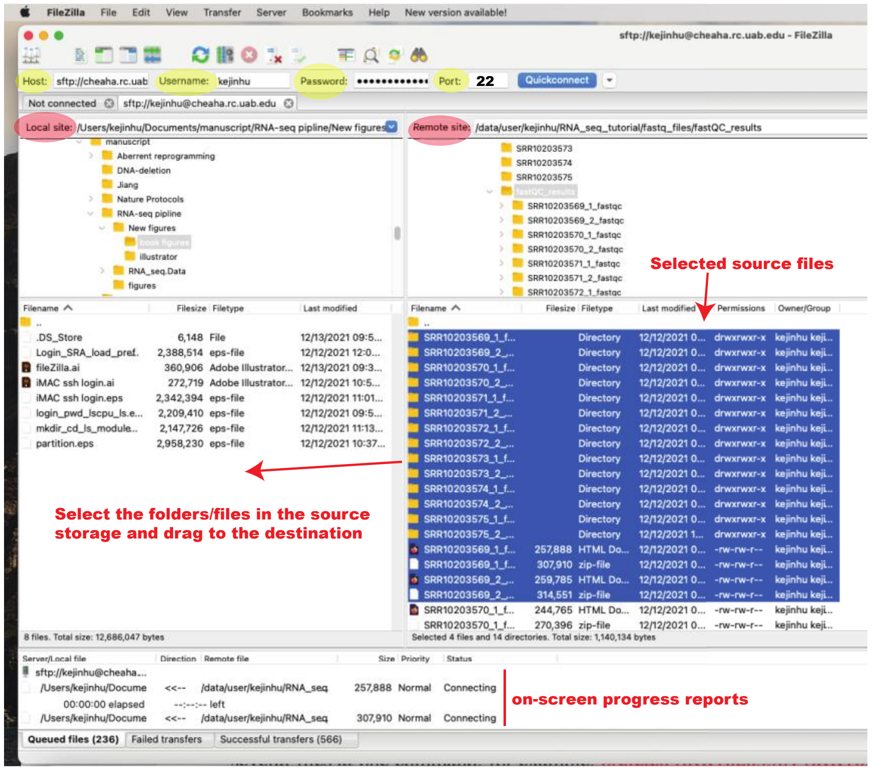
**Figure 10**. Convert the downloaded SRA files to FASTQ files using the command of fasterq-dump on a compute node. Red underline, login node; yellow underline, compute node. Yellow texts are description of scripts or command outputs added by the author.

For the one-step procedure, Cheaha took around the same amount of time to download and convert the 7 SRA files when I requested 10 CPUs and 10 G of memory versus 1 CPU and 1G of memory although the default threading for the fasterq-dump command is 6. In both cases, Cheaha used around 36 minutes to download these 7 RNA-seq from the repository with the command fasterq-dump for one step downloading.

## 2.7 Upload FASTQ files onto HPC by other means

You can also transfer your raw FASTQ data onto HPC using the command wget, rsync, Globus, FileZilla, rclone, or other means depending on where your source FASTQ files are located. The sequencing facility and your HPC platform may prefer a specific type of tool, for example, the Cheaha team recommends Globus. Seek help from the sequencing facility or your HPC team if you need assistance at this step.

Globus and FileZilla are very intuitive and easy to use without the need of coding (or commands). Audiences would take no time to master the technique of data transfer using FileZilla or Globus since those both are graphic interfaces. As an example, Figure 11 depicts how to transfer data between local iMAC and HPC using FileZilla. To connect to your HPC using FileZilla, you just need the host name (HPC as your host in this tutorial), your HPC username, your HPC account password, and the SSH port number (highlighted in yellow shading in Figure 11). The default port number is 22, but it may be redefined. If 22 would not the port number, you can ask your HPC support team. After providing values for these parameters, you just click the Quickconnect button (the blue button in Figure 11) to connect to your HPC (remote site, highlighted in red ellipse in Figure 11). The transfer is very simple. You just select the files/folders from the source storage (the directories and files shaded in blue in Figure 11) and drag them to the destination storage (Figure 11).


FileZilla File Edit View Transfer Server Bookmarks Help New version available!
sftp://kejinhu@cheaha.rc.uab.edu - FileZilla
Quickconnect
Selected source files
Select the folders/files in the source storage and drag to the destination
on-screen progress reports
Queued files (236)
Failed transfers
Successful transfers (566)

Host: sftp://cheaha.rc.uab Username: kejinhu Password: •••••••••••• Port: 22

Not connected | sftp://kejinhu@cheaha.rc.uab.edu

Local site: /Users/kejinhu/Documents/manuscript/RNA-seq pipline/New figures

```
manuscript
  Aberrent reprogramming
  DNA-deletion
  Jiang
  Nature Protocols
  RNA-seq pipline
    New figures
      book figures
      illustrator
    RNA_seq.Data
    figures
```

Remote site: /data/user/kejinhu/RNA_seq_tutorial/fastq_files/fastQC_results

```
SRR10203573
SRR10203574
SRR10203575
fastQC_results
  SRR10203569_1_fastqc
  SRR10203569_2_fastqc
  SRR10203570_1_fastqc
  SRR10203570_2_fastqc
  SRR10203571_1_fastqc
  SRR10203571_2_fastqc
  SRR10203572_1_fastqc
```

| Filename | Filesize | Filetype | Last modified |
|---|---|---|---|
| .. | | | |
| .DS_Store | 6,148 | File | 12/13/2021 09:5... |
| Login_SRA_load_pref. | 2,388,514 | eps-file | 12/12/2021 12:0... |
| fileZilla.ai | 360,906 | Adobe Illustrator... | 12/13/2021 09:3... |
| iMAC ssh login.ai | 272,719 | Adobe Illustrator... | 12/12/2021 10:5... |
| iMAC ssh login.eps | 2,342,394 | eps-file | 12/12/2021 11:01... |
| login_pwd_lscpu_ls.e... | 2,209,410 | eps-file | 12/12/2021 09:5... |
| mkdir_cd_ls_module... | 2,147,726 | eps-file | 12/12/2021 11:13... |
| partition.eps | 2,958,230 | eps-file | 12/12/2021 10:37... |

8 files. Total size: 12,686,047 bytes

| Filename | Filesize | Filetype | Last modified | Permissions | Owner/Group |
|---|---|---|---|---|---|
| .. | | | | | |
| SRR10203569_1_f... | | Directory | 12/12/2021 0... | drwxrwxr-x | kejinhu keji... |
| SRR10203569_2_... | | Directory | 12/12/2021 0... | drwxrwxr-x | kejinhu keji... |
| SRR10203570_1_f... | | Directory | 12/12/2021 0... | drwxrwxr-x | kejinhu keji... |
| SRR10203570_2_... | | Directory | 12/12/2021 0... | drwxrwxr-x | kejinhu keji... |
| SRR10203571_1_f... | | Directory | 12/12/2021 0... | drwxrwxr-x | kejinhu keji... |
| SRR10203571_2_... | | Directory | 12/12/2021 0... | drwxrwxr-x | kejinhu keji... |
| SRR10203572_1_f... | | Directory | 12/12/2021 0... | drwxrwxr-x | kejinhu keji... |
| SRR10203572_2_... | | Directory | 12/12/2021 0... | drwxrwxr-x | kejinhu keji... |
| SRR10203573_1_f... | | Directory | 12/12/2021 0... | drwxrwxr-x | kejinhu keji... |
| SRR10203573_2_... | | Directory | 12/12/2021 0... | drwxrwxr-x | kejinhu keji... |
| SRR10203574_1_f... | | Directory | 12/12/2021 0... | drwxrwxr-x | kejinhu keji... |
| SRR10203574_2_... | | Directory | 12/12/2021 0... | drwxrwxr-x | kejinhu keji... |
| SRR10203575_1_f... | | Directory | 12/12/2021 0... | drwxrwxr-x | kejinhu keji... |
| SRR10203575_2_... | | Directory | 12/12/2021 1... | drwxrwxr-x | kejinhu keji... |
| SRR10203569_1_f... | 257,888 | HTML Do... | 12/12/2021 0... | -rw-rw-r-- | kejinhu keji... |
| SRR10203569_1_f... | 307,910 | zip-file | 12/12/2021 0... | -rw-rw-r-- | kejinhu keji... |
| SRR10203569_2_... | 259,785 | HTML Do... | 12/12/2021 0... | -rw-rw-r-- | kejinhu keji... |
| SRR10203569_2_... | 314,551 | zip-file | 12/12/2021 0... | -rw-rw-r-- | kejinhu keji... |
| SRR10203570_1_f... | 244,765 | HTML Do... | 12/12/2021 0... | -rw-rw-r-- | kejinhu keji... |
| SRR10203570_1_f... | 270,396 | zip-file | 12/12/2021 0... | -rw-rw-r-- | kejinhu keji... |

Selected 4 files and 14 directories. Total size: 1,140,134 bytes

```
Server/Local file        Direction  Remote file                  Size  Priority  Status
sftp://kejinhu@cheaha...
 /Users/kejinhu/Docume   <<--       /data/user/kejinhu/RNA_seq   257,888  Normal  Connecting
      00:00:00 elapsed   --:--:-- left
 /Users/kejinhu/Docume   <<--       /data/user/kejinhu/RNA_seq   307,910  Normal  Connecting
```

**Figure 11**. FileZilla as a tool for data transfer between local (your computer) and remote HPC storage (both shaded in red ellipses). Shaded in yellow are parameters you need to define (host name, username, port, and password). Red texts are the author’s description. Shaded in blue are the folders and files to be transferred.

## Summary of Chapter 2

- The commands module spider and module avail can be used to find out installed modules on HPC with a syntax of module spider [module name]. The module name with these two commands is not case sensitive and can be incomplete.

- The command module load can be used to load a software to run with a syntax of module load [module name]. The module name with this command is case sensitive, and should be given in full except for the part of version number.

- There are different tools for transferring RNA-seq raw data to HPC depending on where your raw data are located.

- SRA is the repository for the RNA-seq raw data, and SRA-Toolkit is the designed software to download SRA data using the prefetch command and then convert the downloaded data into FASTQ files using the fasterq-dump command.

- The SRA files can be downloaded and converted at one step using the fasterq-dump command, but the two-step procedure is faster and more reliable. The data transfer process can be optimized using the multiple threading option of the fasterq-dump command. The default threading (-e or --threads) is 6 CPUs, and 12 CPUs usually take no time to be allocated from a good HPC cluster.

- FileZilla and Globus are two simple interfaces for data transfer between HPC and other storages since those are graphic interfaces.

- Data transfer between HPC and other storages should be done on the compute nodes, not the login nodes.

# Chapter 3

## Request compute resources

## and

## QC RNA-seq data

## 3.1 Conduct QC of RNA-seq data as an interactive job on a compute node

Once you have your RNA-seq FASTQ files in the fastq_files directory of your HPC account, the first thing you do is to examine the quality of the raw RNA-seq data. This can be done using FastQC. FastQC is widely used, and likely available on your HPC. To QC your RNA-seq data in FASTQ format, we can request computational resources on a compute node for interactive tasks using the command srun introduced in Chapter 2 already since QC is not a heavy job.

> srun --pty /bin/bash # This requests the default resources from compute nodes for interactive operation. We will request for more resources with resource options when we run a heavy computing job. In that case, we would better use the sbatch command of SLURM (see below). Please note in the interactive mode on compute nodes requested by srun, you will lose the resources if you close the pseudo terminal using the exit command. For a light job, the default resources may be more than you need. You should exit the compute node when you will no longer use it so that others can use it. You need to use the sbatch command to submit a non-interactive job.

To check if FastQC has been installed on your HPC, use the module spider or module avail command,

> module spider fastqc # This command lists all the versions of FastQC installed on your HPC. The module name here is not case sensitive, which avoids fusses since one is usually not sure about the exact letter case of a module name. The command module avail gives you similar information as module spider does. You can also find the list of FastQC package versions using incomplete package names with these two commands, for examples, module spider fastq, or module avail fast.

Now, you can load the FastQC module/software using the command module load,

> module load FastQC # This loads the FastQC module/software onto the nodes allocated to you. Please note that unlike the commands of module spider and module avail the module name (i.e., FastQC here) is case sensitive when you use the command module load. You can also explicitly load a FastQC version by specifying the entire version information: e.g., module load FastQC/0.11.7-Java-1.8.0.74. You can call module avail fastqc after loading it, and you will see which version has been loaded. The loaded version is indicated by an “L”. When you do not include the version code the default version is loaded. The default version is indicated by a “D” when you output the list of versions of a module using “module avail”.

To find out the module version and retrieve the online help page, issue, fastqc --version # This command lists the version of the loaded FastQC using the option of --version. Please note that "fastqc" here in this command is all in lower case. Otherwise, you will have a message like: "bash: FastQC: command not found" in the case you call "FastQC --version".

fastqc --help # This displays brief information about the FastQC command fastqc using the option of --help. Please note that "fastqc" here in this command is all in lower case because the command fastqc is all in lower case (see below).

A good practice is that you save the QC results in a specified directory, fastQC_results, and this avoid cluttering the fastq_files directory by the many resulting QC files and subdirectories. Make such a subdirectory under the directory of fastq_files using the Make Directory command mkdir,

mkdir fastQC_results # You directly use this command when you are in the fastq_files directory already. If not, use the command pwd to find out your location, and then use the command cd to move to the fastq_files directory. Alternatively, you can provide the relative or full path to the new directory you are making. For example, you can make a subdirectory of test1 inside the newly generated fastQC_results using a relative path, mkdir ./fastQC_results/test1, or a full path, mkdir /data/user/kejinhu/RNA_seq_tutorial/fastq_files/fastQC_results/test1. The ./ (dot slash) here denotes the current directory.

ls # Using the list command ls, you can see the newly generated directory of fastQC_results within the directory of fastq_files.

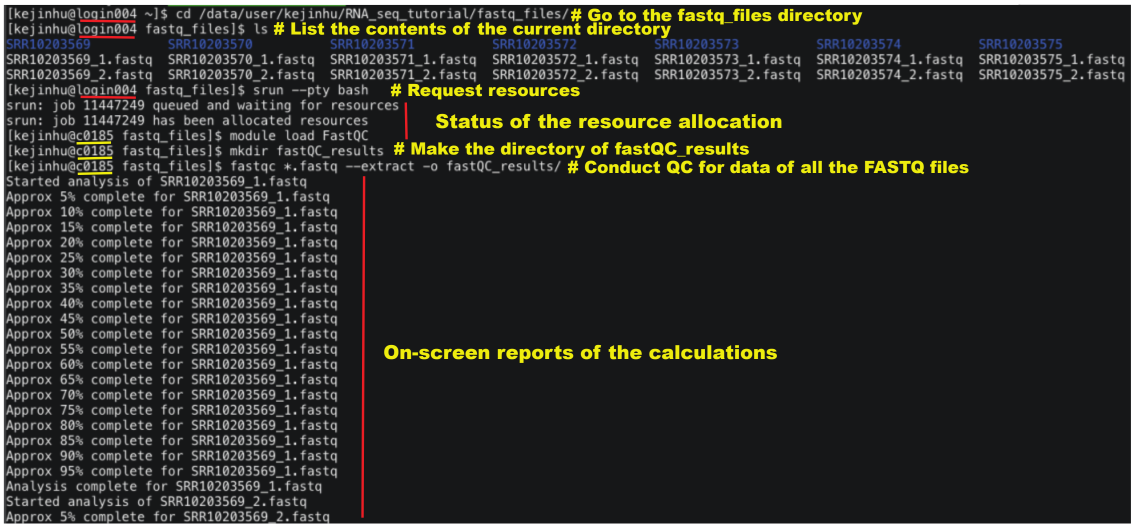
You can QC a single file using the syntax below,

fastqc myRNAseq_file1.fastq --extract -o fastQC_results # e.g., you can QC the file SRR10203569_1.fastq in this tutorial: fastqc SRR10203569_1.fastq --extract -o fastQC_results. For description of the options for the command of fastqc, see comments in the next script below.

You can use a single and short code below to QC all the FASTQ files inside the fastq_files directory, and save the QC results into the subdirectory of fastQC_results,

fastqc *.fastq --extract -o fastQC_results/ # The --extract option has no value and means the zipped output files will be uncompressed in the same directory after they have been created. If you do not use the --extract option you will see the zip and the HTML files only. In this case, you will uncompress the zip files using the command of unzip. In the non-interactive mode, --extact is the default option. The -o option specifies the directory in which the output files will be saved, and it is the one-letter version of the --outdir= option. The output directory for the -o option need to be pre-defined and the fastqc command cannot generate the fastQC_results directory itself. When the output directory is not pre-defined you will see "Specified output directory, 'fastQC_results' does not exist"in the .out file. When you choose the long or word version of the --outdir= option, you use --outdir = fastQC_results instead. You can and may not include / in the option -o /fastQC_results. You can issue the fastqc --help command to see the usage of the --extract and -o options.

When you run the above codes in the interactive mode on a compute node, you may see the computing progress information on your terminal as shown in the Figure 12.



```
[kejinhu@login004 ~]$ cd /data/user/kejinhu/RNA_seq_tutorial/fastq_files/# Go to the fastq_files directory
[kejinhu@login004 fastq_files]$ ls # List the contents of the current directory
SRR10203569        SRR10203570        SRR10203571        SRR10203572        SRR10203573        SRR10203574        SRR10203575
SRR10203569_1.fastq  SRR10203570_1.fastq  SRR10203571_1.fastq  SRR10203572_1.fastq  SRR10203573_1.fastq  SRR10203574_1.fastq  SRR10203575_1.fastq
SRR10203569_2.fastq  SRR10203570_2.fastq  SRR10203571_2.fastq  SRR10203572_2.fastq  SRR10203573_2.fastq  SRR10203574_2.fastq  SRR10203575_2.fastq
[kejinhu@login004 fastq_files]$ srun --pty bash   # Request resources
srun: job 11447249 queued and waiting for resources
srun: job 11447249 has been allocated resources
[kejinhu@c0185 fastq_files]$ module load FastQC
[kejinhu@c0185 fastq_files]$ mkdir fastQC_results # Make the directory of fastQC_results
[kejinhu@c0185 fastq_files]$ fastqc *.fastq --extract -o fastQC_results/ # Conduct QC for data of all the FASTQ files
Started analysis of SRR10203569_1.fastq
Approx 5% complete for SRR10203569_1.fastq
Approx 10% complete for SRR10203569_1.fastq
Approx 15% complete for SRR10203569_1.fastq
Approx 20% complete for SRR10203569_1.fastq
Approx 25% complete for SRR10203569_1.fastq
Approx 30% complete for SRR10203569_1.fastq
Approx 35% complete for SRR10203569_1.fastq
Approx 40% complete for SRR10203569_1.fastq
Approx 45% complete for SRR10203569_1.fastq
Approx 50% complete for SRR10203569_1.fastq
Approx 55% complete for SRR10203569_1.fastq
Approx 60% complete for SRR10203569_1.fastq
Approx 65% complete for SRR10203569_1.fastq
Approx 70% complete for SRR10203569_1.fastq
Approx 75% complete for SRR10203569_1.fastq
Approx 80% complete for SRR10203569_1.fastq
Approx 85% complete for SRR10203569_1.fastq
Approx 90% complete for SRR10203569_1.fastq
Approx 95% complete for SRR10203569_1.fastq
Analysis complete for SRR10203569_1.fastq
Started analysis of SRR10203569_2.fastq
Approx 5% complete for SRR10203569_2.fastq
```

Status of the resource allocation
On-screen reports of the calculations

**Figure 12**. Conduct QC of the RNA-seq raw data using the fastqc command of the FastQC software. The job is conducted in interactive mode established using srun.

## 3.2. Submit QC work as a non-interactive job using the sbatch command

When you submit a job to the compute nodes using the srun command, you usually stay on the computing screen until the computing is completed. It is not convenient when the process takes a long time; or you conduct the same operation for many samples (repetitive jobs). We can submit a long or repetitive job using the sbatch command and immediately work on other light tasks on the login node, or you can even shut down your desktop after submission of a non-interactive job. To do this, we make a SLURM script file first and subsequently submit the file using the following simple syntax,

sbatch my-slurm-script-file-name.slurm        # The job script file usually uses the extension of .sh, standing for bash. In this tutorial, the author prefers the file extension of .slurm since it is more informative.

Using the QC computing as an example, we can prepare a SLURM script in the Linux nano text editor. Open nano editor with a file name of fastQC-RNAseq.slurm (Figures 13 and 14),

> nano fastQC-RNAseq.slurm     # This takes you to the nano text editor, and you can prepare your script there. For the usage of nano text editor, you can use the Linux manual function (man), man nano; or audiences can read the online tutorial about it (https://www.nano-editor.org/ . You can find the online manual or the PDF version via the Documentation tab on the nano home page).

In the nano text editor, prepare the following script or a similar one (Figure 13),

```
#!/bin/bash
#SBATCH --time=02:00:00
#SBATCH --partition=express
#SBATCH --nodes=1
#SBATCH --cpus-per-task=1
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=5G

module load FastQC

fastqc /data/user/kejinhu/RNA_seq_tutorial/fastq_files/*.fastq --extract \
-o /data/user/kejinhu/RNA_seq_tutorial/fastq_files/fastQC_results
```

There are two basic parts in the bash script above (also see Figure 13). First, we should specify the resources needed for the non-interactive job. This is called the SLURM directives section. The resource options include partition category, number of tasks, number of nodes, number of CPUs, wall time, and minimum memory for each CPU (--cpus-per-task) or for the job (--mem=). In the above code we request 1 CPU within one node, 5 G of memory per CPU with the *short* partition for up to 2 hours of time for 1 task. The SLURM directives section may define other auxiliary features of the file, for examples, options of --mail-type=, --mail-user=, --error=, --out=, --chdir=, --job-name=, and others (refer to Sections 4.3 and 4.4, and Figures 23 and 25).

Second, you need to write codes for the specific computation (in the current case, codes for QC of the RNA-seq data in FASTQ format using the software of FastQC). This second part is called the job command section. Please note that you need to include a code for loading the software/module before the application codes using the command of module load (see codes above and in Figure 13). If you want a command line to continue in a different line, you can use the backslash operator \ to break a long command line into 2 lines as used in the above code for the fastqc command (in Figure 13) (see more notes about using a long command line in nano text editor in Section 4.3).

After the script is written, you can save it as instructed at the bottom of the nano text editor, i.e., using the key combination of "control" and "O" (bottom of Figure 23). Then you can exit the nano text editor using the key combination of "control" and "X" (bottom of Figure 23).

In the directory the script file is located (i.e., fastq_files directory here), submit the job using the sbatch command (Figure 14),

> sbatch fastQC-RNAseq.slurm # Please note that this code submits the job under the directory where the fastQC-RNAseq.slurm file is located. If you submit it from any other directory, you should include the path to this script file. The SBATCH script file could be in a directory different from the FASTQ data directory. When you put and submit the SLURM script file in the directory where the FASTQ files are located as we do here, you can shorten the paths for both the input and output files in the bash script as, fastqc ./*.fastq --extract -o ./fastQC_results. The dot "." denotes "current directory", which is the directory the script file fastQC-RNAseq.slurm is submitted. However, if your script file is in another directory, you should not use this short code and need to specify the full paths. Please note that the working directory is where you submit the SBATCH script file not where it is located. Using the above code, it took Cheaha around 29 minutes to complete the QC process for the 14 FASTQ files in this tutorial. There was not much improvement when the CPU was increased to 14 (i.e., --cpus-per-task=14) because fastqc still uses one CPU when the --thread option is not defined (see next section below for multiple threading). The working directory of the SBATCH script file can be defined using the option of --chdir (see Section 4.3 and Figure 23). When defined by the --chdir option, the working directory is not affected by where you submit the script file.

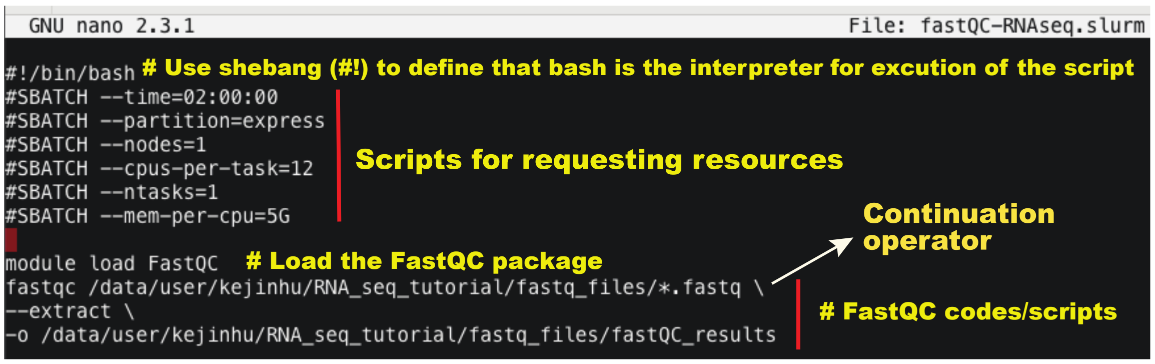
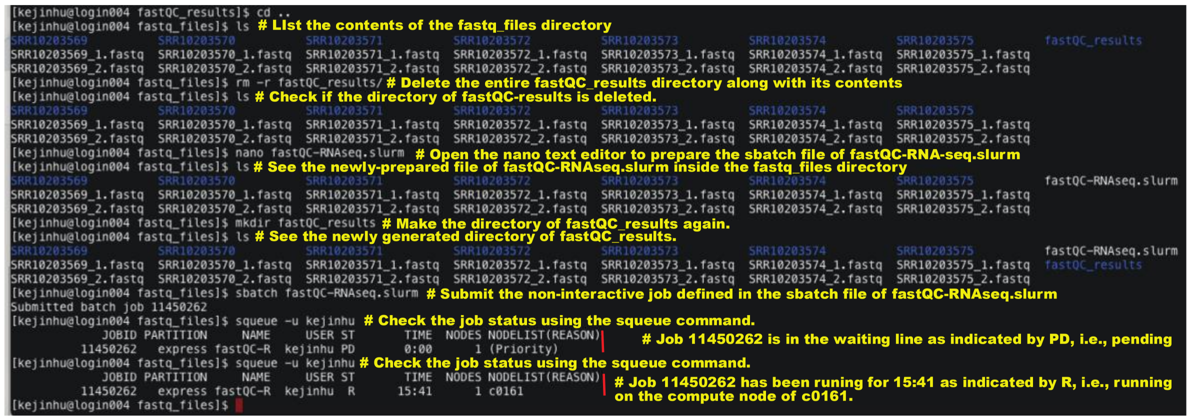

```
  GNU nano 2.3.1                                                File: fastQC-RNAseq.slurm

#!/bin/bash # Use shebang (#!) to define that bash is the interpreter for excution of the script
#SBATCH --time=02:00:00
#SBATCH --partition=express
#SBATCH --nodes=1
#SBATCH --cpus-per-task=12
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=5G

module load FastQC   # Load the FastQC package
fastqc /data/user/kejinhu/RNA_seq_tutorial/fastq_files/*.fastq \
--extract \
-o /data/user/kejinhu/RNA_seq_tutorial/fastq_files/fastQC_results
```

**Figure 13.** Write scripts in the nano text editor for a SBATCH file. This example is a SBATCH file for the fastqc command. The file name is displayed on the upper-right corner of the nano text editor. Please note that the number of CPUs is not optimized in this code and only one CPU was used for QC with this fastqc code (see text for detail). Refer to Figure 23 as well.

## 3.3 Speed up QC using the multiple threading option -t of the fastqc command

The above code to QC the RNA-seq with the FASTQ files uses one CPU only even though you request 12 CPUs. The fastqc command allows multiple threading with the option of --thread or -t. In this tutorial, we have 14 FASTQ files, and therefore we give the -t option a value of 14 and request 14 CPUs. With this new code, Cheaha spent 2 minutes and 53 seconds only to QC the 14 FASTQ files. Please note that you need to request 14 CPUs when you choose --thread=14 for the fastqc command. Otherwise, there is no improvement in speed. Please also note that unlike the multiple threading for the STAR command, the multiple threading for the fastqc command is at the sample levels that means the 14 FASTQC files are processed simultaneously. For the STAR multiple threading, it is at the data level which means the data of each sample are divided into multiple portions and are processed simultaneously (see Chapters 4 and 5).



```
[kejinhu@login004 fastQC_results]$ cd ..
[kejinhu@login004 fastq_files]$ ls # LIst the contents of the fastq_files directory
SRR10203569          SRR10203570          SRR10203571          SRR10203572          SRR10203573          SRR10203574          SRR10203575          fastQC_results
SRR10203569_1.fastq  SRR10203570_1.fastq  SRR10203571_1.fastq  SRR10203572_1.fastq  SRR10203573_1.fastq  SRR10203574_1.fastq  SRR10203575_1.fastq
SRR10203569_2.fastq  SRR10203570_2.fastq  SRR10203571_2.fastq  SRR10203572_2.fastq  SRR10203573_2.fastq  SRR10203574_2.fastq  SRR10203575_2.fastq
[kejinhu@login004 fastq_files]$ rm -r fastQC_results/ # Delete the entire fastQC_results directory along with its contents
[kejinhu@login004 fastq_files]$ ls # Check if the directory of fastQC-results is deleted.
SRR10203569          SRR10203570          SRR10203571          SRR10203572          SRR10203573          SRR10203574          SRR10203575
SRR10203569_1.fastq  SRR10203570_1.fastq  SRR10203571_1.fastq  SRR10203572_1.fastq  SRR10203573_1.fastq  SRR10203574_1.fastq  SRR10203575_1.fastq
SRR10203569_2.fastq  SRR10203570_2.fastq  SRR10203571_2.fastq  SRR10203572_2.fastq  SRR10203573_2.fastq  SRR10203574_2.fastq  SRR10203575_2.fastq
[kejinhu@login004 fastq_files]$ nano fastQC-RNAseq.slurm  # Open the nano text editor to prepare the sbatch file of fastQC-RNA-seq.slurm
[kejinhu@login004 fastq_files]$ ls # See the newly-prepared file of fastQC-RNAseq.slurm inside the fastq_files directory
SRR10203569          SRR10203570          SRR10203571          SRR10203572          SRR10203573          SRR10203574          SRR10203575          fastQC-RNAseq.slurm
SRR10203569_1.fastq  SRR10203570_1.fastq  SRR10203571_1.fastq  SRR10203572_1.fastq  SRR10203573_1.fastq  SRR10203574_1.fastq  SRR10203575_1.fastq
SRR10203569_2.fastq  SRR10203570_2.fastq  SRR10203571_2.fastq  SRR10203572_2.fastq  SRR10203573_2.fastq  SRR10203574_2.fastq  SRR10203575_2.fastq
[kejinhu@login004 fastq_files]$ mkdir fastQC_results # Make the directory of fastQC_results again.
[kejinhu@login004 fastq_files]$ ls # See the newly generated directory of fastQC_results.
SRR10203569          SRR10203570          SRR10203571          SRR10203572          SRR10203573          SRR10203574          SRR10203575          fastQC-RNAseq.slurm
SRR10203569_1.fastq  SRR10203570_1.fastq  SRR10203571_1.fastq  SRR10203572_1.fastq  SRR10203573_1.fastq  SRR10203574_1.fastq  SRR10203575_1.fastq  fastQC_results
SRR10203569_2.fastq  SRR10203570_2.fastq  SRR10203571_2.fastq  SRR10203572_2.fastq  SRR10203573_2.fastq  SRR10203574_2.fastq  SRR10203575_2.fastq
[kejinhu@login004 fastq_files]$ sbatch fastQC-RNAseq.slurm # Submit the non-interactive job defined in the sbatch file of fastQC-RNAseq.slurm
Submitted batch job 11450262
[kejinhu@login004 fastq_files]$ squeue -u kejinhu # Check the job status using the squeue command.
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)      # Job 11450262 is in the waiting line as indicated by PD, i.e., pending
          11450262   express fastQC-R  kejinhu PD       0:00      1 (Priority)
[kejinhu@login004 fastq_files]$ squeue -u kejinhu # Check the job status using the squeue command.
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)  # Job 11450262 has been runing for 15:41 as indicated by R, i.e., running
          11450262   express fastQC-R  kejinhu  R      15:41      1 c0161               on the compute node of c0161.
[kejinhu@login004 fastq_files]$ 
```

**Figure 14** Submit a sbatch job, and check the job status using the squeue command with the -u option. The yellow texts are the author's illustration.

## 3.4 Submit your computation work (QC job as an example) via the web Job Composer

Your HPC may have a job composer on its web portal, which provides a graphical interface to your HPC. You may make a SLURM job request for your computation project just by clicking the buttons and tabs (Figures 15-17). You can also write your batch script using the associated more user-friendly web text editor (Figure 17). Here, we use FastQC as an example to introduce how to submit non-interactive jobs via the web job composer powered by the OSC Open OnDemand.

On your HPC web dashboard, extend the pulldown menu of the Jobs tab (Figure 2) and click on the Job Composer button on the menu to open the Job Composer interface (Figures 15 and 16). On Job Composer page, find the text editor pane and click the Open Editor (cyan button in Figure 16) to get access to the job text editor (Figure 17). Compose the script and click the Save button (Figure 17). When you are back to the job composer main page, click the Submit button, and you will see that your job statuses change from Not Submitted, to Queued, to Running, and finally to Completed when the process will be finished (Figure 15).

During the computation you can go to the HPC terminal and check the status using, squeue -u [your HPC username] (lower part in Figure 14). After the computation you can see the resulting files using the ls command under the destination directory. Apparently, you can click some buttons at some steps of the job submission processes instead of using command lines, but you may find that using the command lines is more convenient when you become familiar with just some basic commands of SLURM.

In the web Job Composer, the sbatch script is similar to that in srun and that of the traditional sbatch script introduced above. In the FastQC code part, the only difference of the SLURM code from that of the login mode or interactive compute mode is that you need to define the full path to the FASTQ files to be analyzed. You should define the path to the output file directory as well. This is because the script file is located in and submitted from the myjobs directory of the home directory by default, which is the working directory. However, you can define the working directory using the --chdir option in the SBATCH directives section.

## 3.5 Review the FastQC results

For each FASTQ file, you will have one subdirectory generated automatically, one compressed file, and one HTML file. Go to the subdirectory for one of the FastQC result files using the cd command. You will see that the associated subdirectory contains 4 files and 2 subsubdirectories. The 4 files are: fastqc_data.txt, fastqc.fo, fastqc_report.html, and summary.txt. The two subdirectories are Icons and Images. Both directories contain many png files.

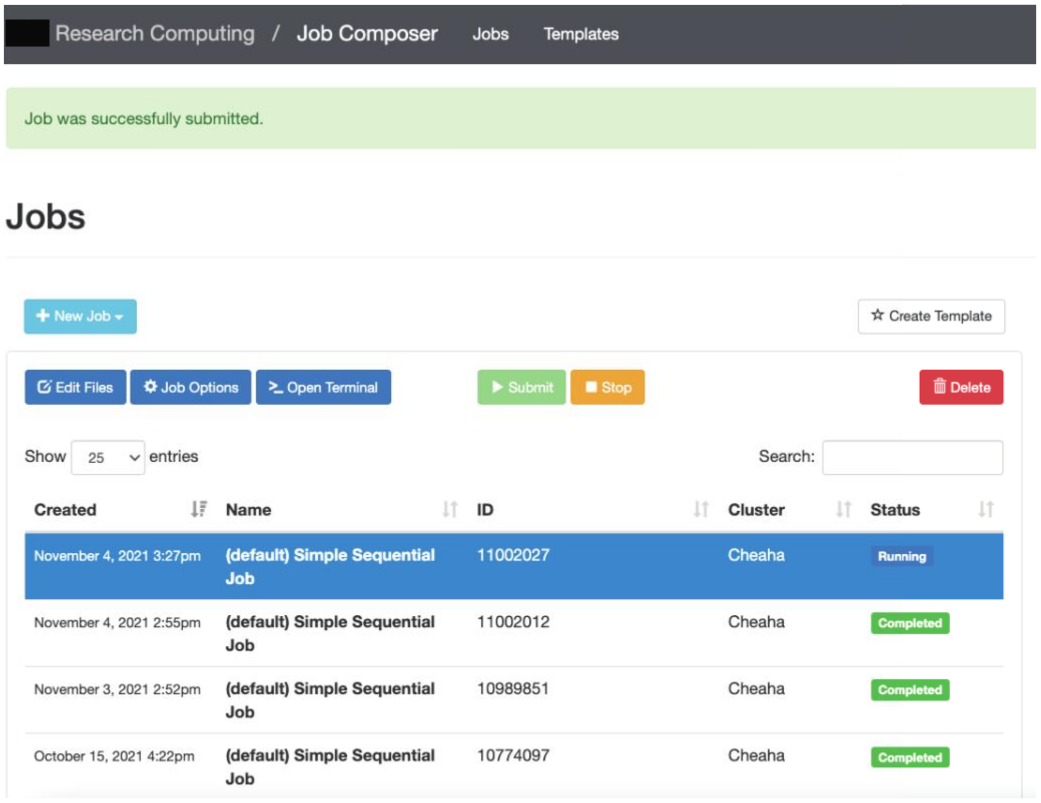
### 3.5.1 Review the FastQC results using the nano text editor, or the generic Linux commands

To do this, go to the FastQC result directory of an RNA-seq using cd, and use the nano command,

nano summary.txt      # You can examine the results using the cat utility, i.e., cat summary.txt, but the nano text editor displays the output in a way that is easy to read. Please note that you need to open the summary.txt file with the nano text editor in the directory where the file summary.txt is located. Otherwise, you will open the nano text editor with an empty workspace. If you do want to open a file using the nano command from a different directory, you should include the path to the file, i.e., nano path_to_the_file/fine_name. For the usage of nano text editor, you can use the Linux manual command, man nano, or read online tutorial about it (https://www.nano-editor.org/ , you can find the online manual or the PDF version via the Documentation tab of the nano home page).

You may see results like,

```
PASS    Basic Statistics                SRR10203569_1.fastq
PASS    Per base sequence quality        SRR10203569_1.fastq
PASS    Per tile sequence quality       SRR10203569_1.fastq
PASS    Per sequence quality scores     SRR10203569_1.fastq
FAIL    Per base sequence content       SRR10203569_1.fastq
PASS    Per sequence GC content         SRR10203569_1.fastq
PASS    Per base N content              SRR10203569_1.fastq
PASS    Sequence Length Distribution    SRR10203569_1.fastq
FAIL    Sequence Duplication Levels     SRR10203569_1.fastq
PASS    Overrepresented sequences       SRR10203569_1.fastq
PASS    Adapter Content                 SRR10203569_1.fastq
```

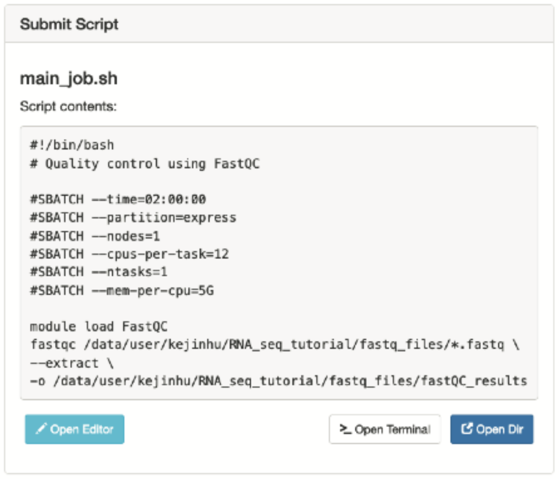

[redacted] Research Computing / Job Composer   Jobs   Templates

Job was successfully submitted.

Jobs

| Created | Name | ID | Cluster | Status |
|---|---|---|---|---|
| November 4, 2021 3:27pm | (default) Simple Sequential Job | 11002027 | Cheaha | Running |
| November 4, 2021 2:55pm | (default) Simple Sequential Job | 11002012 | Cheaha | Completed |
| November 3, 2021 2:52pm | (default) Simple Sequential Job | 10989851 | Cheaha | Completed |
| October 15, 2021 4:22pm | (default) Simple Sequential Job | 10774097 | Cheaha | Completed |

**Figure 15**. Open OnDemand-based Web interface for SLURM job submission. The screenshot did not capture all the features. As you can see some processes can be done by clicking a button, for example, submitting a job by clicking Submit instead of using the sbatch command. But the web interface is not necessarily convenient for an experienced user of Linux and SLURM.


Submit Script

main_job.sh

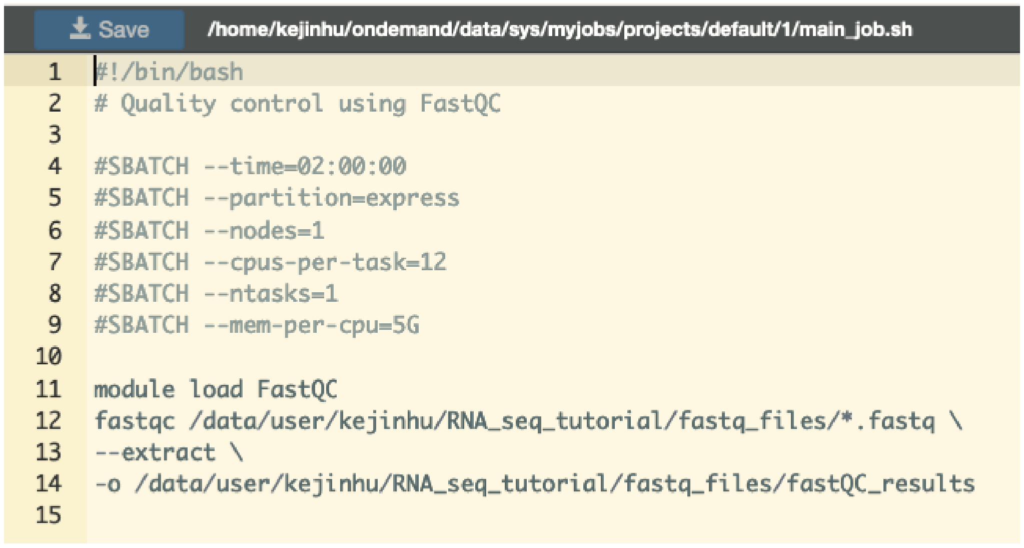
Script contents:

```
#!/bin/bash
# Quality control using FastQC

#SBATCH --time=02:00:00
#SBATCH --partition=express
#SBATCH --nodes=1
#SBATCH --cpus-per-task=12
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=5G

module load FastQC
fastqc /data/user/kejinhu/RNA_seq_tutorial/fastq_files/*.fastq \
--extract \
-o /data/user/kejinhu/RNA_seq_tutorial/fastq_files/fastQC_results
```

**Figure 16**. The web composer interface showing the section for opening the web text editor by clicking the Open Editor button (cyan button on the lower left) rather than using the nano file-name command. The script contents for the selected job (the job shaded in blue in Figure 15) can be seen here.

You can also review the fastqc_data.txt file in the nano text editor using the nano command,

nano fastqc_data.txt # The content of this file is long, and you can page down and up to review it.


Save
/home/kejinhu/ondemand/data/sys/myjobs/projects/default/1/main_job.sh

```
#!/bin/bash
# Quality control using FastQC

#SBATCH --time=02:00:00
#SBATCH --partition=express
#SBATCH --nodes=1
#SBATCH --cpus-per-task=12
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=5G

module load FastQC
fastqc /data/user/kejinhu/RNA_seq_tutorial/fastq_files/*.fastq \
--extract \
-o /data/user/kejinhu/RNA_seq_tutorial/fastq_files/fastQC_results
```

**Figure 17**. OnDemand web text editor. Captured is part of the text editor with the fastqc codes and the **Save** button.

Alternatively, you can check the fastqc_data.txt file using the less command,

less fastqc_data.txt # You can use the up- or down-arrow keys, or page up and page down keys to navigate along the text. To exit the text file in the less mode, hit the q key. The cat command can also output the file contents on the screen (try cat fastqc_data.txt), but the less command is better since this file is very long.

### 3.5.2 Review the FastQC results on HPC virtual desktop

The most efficient way to review the QC results is using the HTML files, which will give you the graphical summaries of the QC analyses. To do this, we quickly launch the HPC virtual desktop from Cheaha OnDemand web portal (via the "Interactive Apps" tab in Figure 2). On the HPC desktop, open the terminal emulator. Then, go to the directory that contains the HTML file using the cd command, and run,

```
module spider firefox

module load firefox
```

firefox sampleName.fastqc.html # You will see the FastQC results on the Firefox browser after running this code.

In the simplest way, you just go to the directory of the fastQC_results via the Windows-style File Manager on the HPC desktop interface, and then click on the HTML file name or icon.

#### 3.5.3 Review the FastQC results on your desktop browser

Alternatively, there is still a simple and familar way to see the QC results using the generated HTML files if your HPC does not have a HPC desktop interface. You can just transfer the HTML files onto your desktop hard drive using FileZilla or Globus. Then, you can open it simply by clicking on one file name of the QC HTML files.

### 3.6 Aggregate the FastQC results of your projects using the MultiQC facility

Using FastQC, you get results for each RNA-seq FASTQ file. It is tedious to examine all the samples one by one when your projects involve a lot of samples. MultiQC is a tool that generates one HTML file aggregating all the QC results based on FastQC. The syntax is very simple.

srun --pty bash # This will request compute resources using the default options.

module spider multiqc # Or, you can just use module spider multi, or module avail multi.

module load MultiQC

cd fastQC_results # Please note that you should use multiqc command to aggregate the QC results within the fastQC_results directory where the compressed FastQC result files are located. If not in this directory, you can use cd to navigate into this directory.

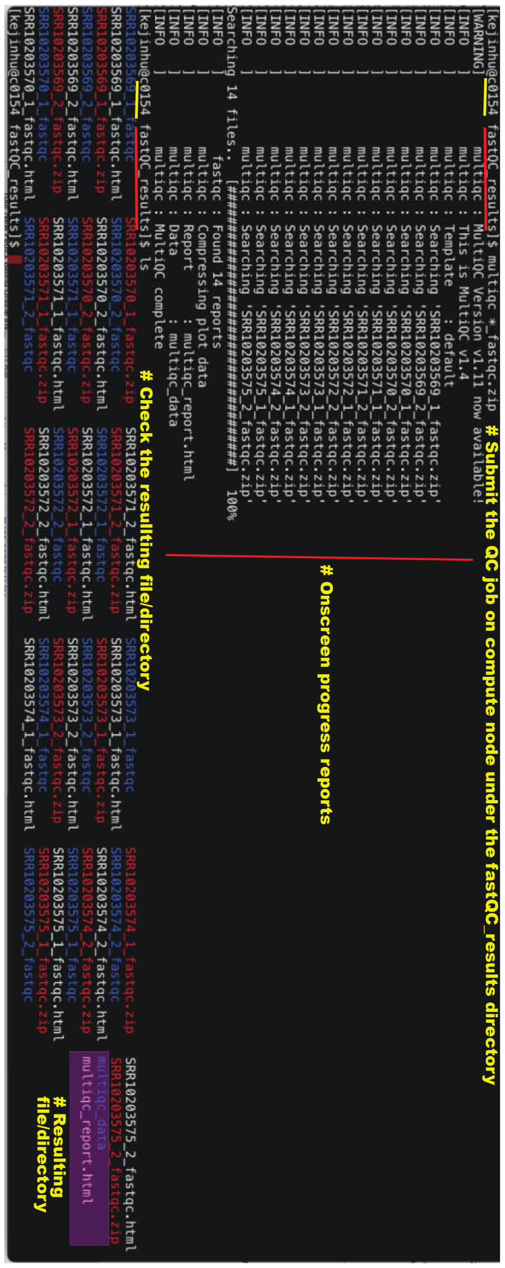
multiqc *_fastqc.zip # This code will aggregate all 14 FastQC result files into one. We simply use the wild card * here to include all QC result files in the current directory without the need to list all the individual files for the multiqc command (Figure 18).

ls # This lists the files in the working directory, and you will see a new file in this directory named multiqc_report.html, and a subdirectory named multiqc_data (Figure 18).

You can also use the multiqc command in the immediate parent directory of the fastQC_results directory containing the fastqc.zip files. In this case the resulting multiqc_data directory and the multiqc_report.html file will be located in the parental directory,

```
multiqc fastQC_results            # In this code, fastQC_results is the directory
containing all the FastQC result zip files.
```

Now, you can examine the aggregated QC results with the file of multiqc_report.html using the methods introduced above for the individual samples.



```
[kejinhu@c0154 fastQC_results]$ multiqc *_fastqc.zip
[WARNING]         multiqc : MultiQC Version v1.11 now available!
[INFO   ]         multiqc : This is MultiQC v1.4
[INFO   ]         multiqc : Template    : default
[INFO   ]         multiqc : Searching 'SRR10203569_1_fastqc.zip'
[INFO   ]         multiqc : Searching 'SRR10203569_2_fastqc.zip'
[INFO   ]         multiqc : Searching 'SRR10203570_1_fastqc.zip'
[INFO   ]         multiqc : Searching 'SRR10203570_2_fastqc.zip'
[INFO   ]         multiqc : Searching 'SRR10203571_1_fastqc.zip'
[INFO   ]         multiqc : Searching 'SRR10203571_2_fastqc.zip'
[INFO   ]         multiqc : Searching 'SRR10203572_1_fastqc.zip'
[INFO   ]         multiqc : Searching 'SRR10203572_2_fastqc.zip'
[INFO   ]         multiqc : Searching 'SRR10203573_1_fastqc.zip'
[INFO   ]         multiqc : Searching 'SRR10203573_2_fastqc.zip'
[INFO   ]         multiqc : Searching 'SRR10203574_1_fastqc.zip'
[INFO   ]         multiqc : Searching 'SRR10203574_2_fastqc.zip'
[INFO   ]         multiqc : Searching 'SRR10203575_1_fastqc.zip'
[INFO   ]         multiqc : Searching 'SRR10203575_2_fastqc.zip'
Searching 14 files..  [####################################]  100%
[INFO   ]          fastqc : Found 14 reports
[INFO   ]         multiqc : Compressing plot data
[INFO   ]         multiqc : Report      : multiqc_report.html
[INFO   ]         multiqc : Data        : multiqc_data
[INFO   ]         multiqc : MultiQC complete
[kejinhu@c0154 fastQC_results]$ ls
SRR10203569_1_fastqc       SRR10203570_1_fastqc.zip   SRR10203571_2_fastqc.html  SRR10203573_1_fastqc       SRR10203574_1_fastqc.zip   SRR10203575_2_fastqc.html
SRR10203569_1_fastqc.html  SRR10203570_2_fastqc       SRR10203571_2_fastqc.zip   SRR10203573_1_fastqc.html  SRR10203574_2_fastqc       SRR10203575_2_fastqc.zip
SRR10203569_1_fastqc.zip   SRR10203570_2_fastqc.html  SRR10203572_1_fastqc       SRR10203573_1_fastqc.zip   SRR10203574_2_fastqc.html  multiqc_data
SRR10203569_2_fastqc       SRR10203570_2_fastqc.zip   SRR10203572_1_fastqc.html  SRR10203573_2_fastqc       SRR10203574_2_fastqc.zip   multiqc_report.html
SRR10203569_2_fastqc.html  SRR10203571_1_fastqc       SRR10203572_1_fastqc.zip   SRR10203573_2_fastqc.html  SRR10203575_1_fastqc
SRR10203569_2_fastqc.zip   SRR10203571_1_fastqc.html  SRR10203572_2_fastqc       SRR10203573_2_fastqc.zip   SRR10203575_1_fastqc.html
SRR10203570_1_fastqc       SRR10203571_1_fastqc.zip   SRR10203572_2_fastqc.html  SRR10203574_1_fastqc       SRR10203575_1_fastqc.zip
SRR10203570_1_fastqc.html  SRR10203571_2_fastqc       SRR10203572_2_fastqc.zip   SRR10203574_1_fastqc.html  SRR10203575_2_fastqc
[kejinhu@c0154 fastQC_results]$
```

# Submit the QC job on compute node under the fastQC_results directory
# Onscreen progress reports
# Check the resullting file/directory
# Resulting file/directory

**Figure 18**. Aggregate the QC results using the multiqc command on a compute node. The compute node is underlined in yellow; the working directory is underlined in red; the resulting QC file and directory are shaded (lower right).

## Summary of Chapter 3

- For any job that takes more than 1 second to complete, you should do it on compute nodes not the login nodes.

- Interactive compute node resources can be requested using the srun command.

- A non-interactive job can be submitted to compute nodes using the sbatch command.

- A computation job can also be submitted using the web composer.

- FastQC can be used to summarize the quality of the RNA-seq raw data, but the results are outputted as individual files for each FASTQ file.

- FastQC allows multiple threading, and such parallel calculation can significantly reduce the operation time.

- MultiQC can aggregate the FastQC results and summarize the entire RNA-seq project in one individual graph or file.

- The text file of the QC results can be reviewed using the Linux commands, cat, or less or the Linux text editor nano. The HTML file of the QC results can be reviewed using a web browser.

# Chapter 4

## Generate reference genome indices

With the high-quality FASTQ data on HPC, the first critical and foundational step in the RNA-seq analysis pipeline is aligning the RNA-seq reads or fragments onto the reference genome. There are different aligners for RNA-seq data. This protocol will provide major procedures for RNA-seq alignment to the human reference genome using the ultra-fast Spliced Transcripts Alignment to a Reference (STAR) aligner, which outperforms other RNA-seq aligners [3, 4]. This tutorial functions as a primer and presents the basic alignment skills. For extensive description of STAR alignment, audiences are referred to the protocols authored by the software developers [4].

Before alignment, we need to establish the human genome indices, which is prepared once only and can be used for all alignments before a new release will be available or when your sequencing lengths are different. For preparation of reference genome indices of other species, the procedures below can be followed similarly.

To generate human genome indices, we need the FASTA and annotation GTF files and the following two sections will walk you through the process for transferring human genome FASTA and GTF files onto HPC from ENSEMBL.

## 4.1 Download and unzip the annotation GTF file of human genome from ENSEMBL

We will put the GTF and FASTA files in a specified directory, humanGenomeIndex under the directory of /data/user/kejinhu/RNA_seq_tutorial. We make this directory first,

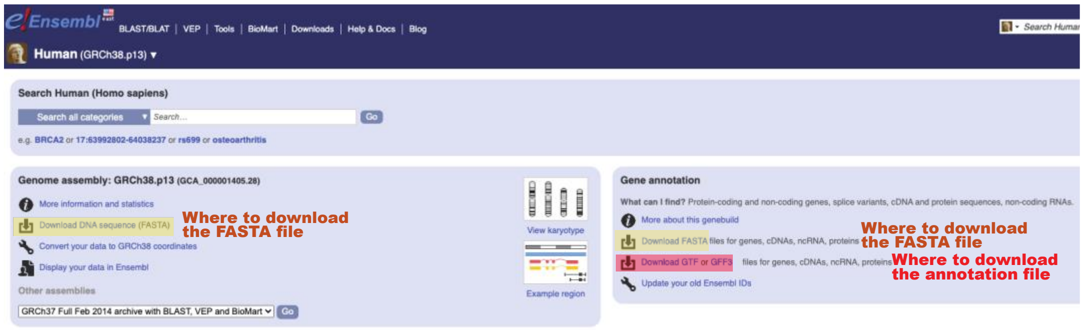
```
cd /data/user/kejinhu/RNA_seq_tutorial

mkdir humanGenomeIndex      # or, you simply do it at one step by providing the
path, mkdir /data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex.

cd  humanGenomeIndex
```

Now, download the GTF file into the humanGenomeIndex directory using the wget command. Go to the ENSEML website (https://useast.ensembl.org/index.html), click on the HTML text "Human" or the passport photo of the Michelangelo's David. On the human ENSEMBL home page, click on the HTML text "download GTF" on the "Gene annotation" panel (Figure 19). On the index page (Figure 20), right click on the HTML text: Homo_sapiens.GRCh38.105.gtf.gz (the version number may not be 105 when you are using this tutorial), and on the popup menu click on "Copy Link Address" (Figure 20).

Now, go back to your HPC terminal and request compute resources. Under the humanGenomeIndex directory type wget and paste the link after the wget command, and then hit the Enter key.

srun --pty /bin/bash # This requests default compute resources from your cluster. In fact, the downloading takes less than 1 minute only.

wget http://ftp.ensembl.org/pub/release-105/gtf/homo_sapiens/Homo_sapiens.GRCh38.105.gtf.gz # Please note that your version number may not be 105 at the time you will use this tutorial. The download is very fast, and just took 6.7 seconds. Of course, you can just download the zipped file to your desktop by clicking the HTML text in Figure 20, and then transfer the file onto HPC using FileZilla or Globus introduced above. But, direct downloading onto HPC using wget is easier and more convenient.



Ensembl BLAST/BLAT | VEP | Tools | BioMart | Downloads | Help & Docs | Blog

Search Human

Human (GRCh38.p13)

Search Human (Homo sapiens)

Search all categories Search... Go

e.g. BRCA2 or 17:63992802-64038237 or rs699 or osteoarthritis

Genome assembly: GRCh38.p13 (GCA_000001405.28)

More information and statistics

Download DNA sequence (FASTA)

Convert your data to GRCh38 coordinates

Display your data in Ensembl

Other assemblies

GRCh37 Full Feb 2014 archive with BLAST, VEP and BioMart Go

View karyotype

Example region

Gene annotation

What can I find? Protein-coding and non-coding genes, splice variants, cDNA and protein sequences, non-coding RNAs.

More about this genebuild

Download FASTA files for genes, cDNAs, ncRNA, proteins

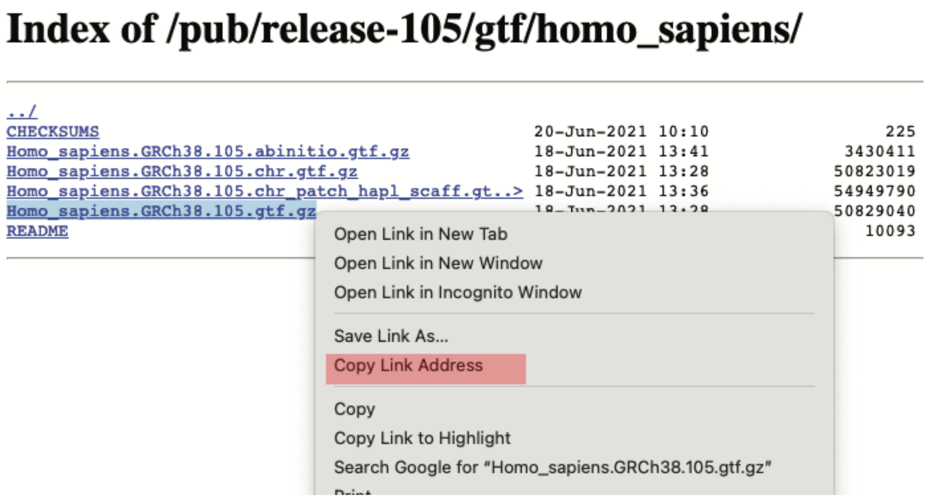
Download GTF or GFF3 files for genes, cDNAs, ncRNA, proteins

Update your old Ensembl IDs

Where to download the FASTA file
Where to download the FASTA file
Where to download the annotation file

**Figure 19.** ENSEMBL hyperlinks for downloading GTF and FASTA files. The respective hyperlinks are shaded and annotated by the author.

We use the .gtf file not the chr.gtf file here (see Figure 20). The chr.gtf file contains annotation for the assembled chromosomes only, and do not include the unplaced or unlocalized contigs. This agrees with the FASTA file we will download later, which



Index of /pub/release-105/gtf/homo_sapiens/

```
../
CHECKSUMS                                          20-Jun-2021 10:10              225
Homo_sapiens.GRCh38.105.abinitio.gtf.gz            18-Jun-2021 13:41          3430411
Homo_sapiens.GRCh38.105.chr.gtf.gz                 18-Jun-2021 13:28         50823019
Homo_sapiens.GRCh38.105.chr_patch_hapl_scaff.gt..> 18-Jun-2021 13:36         54949790
Homo_sapiens.GRCh38.105.gtf.gz                     18-Jun-2021 13:28         50829040
README                                                                          10093
```

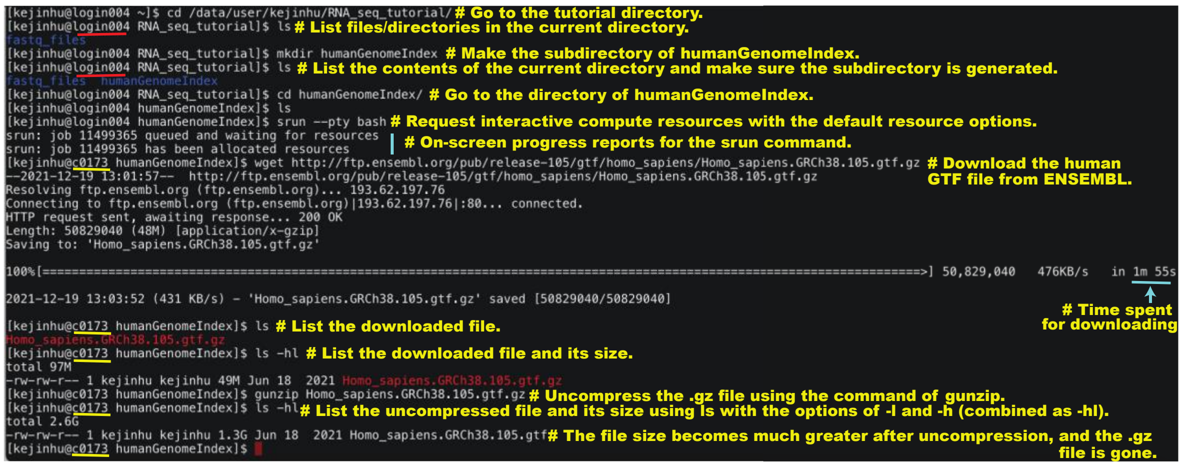
**Figure 20.** Snapshot for downloading human genome GTF file from ENSEMBL. Captured also is the popup window menu for the HTML text of "Homo_sapiens.GRCh38.105.gft.gz". Shaded in red is where the address can be copied onto clipboard by a click.

include the unlocalized contigs as well. Including the unlocalized contigs will reduce the counts for the ‘unmapped reads” and also avoids wrong alignment (see below).

Now, use the ls command to see if the compressed file is there. You should see a file of “Homo_sapiens.GRCh38.105.gtf.gz” in the directory of humanGenomeIndex if the downloading is successful. This file is only 49 MB (Figure 21). Now, uncompress the compressed file using the command of gunzip,

```
gunzip Homo_sapiens.GRCh38.105.gtf.gz
```

After uncompressing, use the ls command and you can see the file of “Homo_sapiens.GRCh38.102.gtf” (Figure 21). Please note that “.gz” is missing because the file is uncompressed now. This file has a size of 1.3 gigabytes if you check with ls -hl (Figure 21).



```
[kejinhu@login004 ~]$ cd /data/user/kejinhu/RNA_seq_tutorial/ # Go to the tutorial directory.
[kejinhu@login004 RNA_seq_tutorial]$ ls # List files/directories in the current directory.
fastq_files
[kejinhu@login004 RNA_seq_tutorial]$ mkdir humanGenomeIndex # Make the subdirectory of humanGenomeIndex.
[kejinhu@login004 RNA_seq_tutorial]$ ls # List the contents of the current directory and make sure the subdirectory is generated.
fastq_files  humanGenomeIndex
[kejinhu@login004 RNA_seq_tutorial]$ cd humanGenomeIndex/ # Go to the directory of humanGenomeIndex.
[kejinhu@login004 humanGenomeIndex]$ ls
[kejinhu@login004 humanGenomeIndex]$ srun --pty bash # Request interactive compute resources with the default resource options.
srun: job 11499365 queued and waiting for resources  | # On-screen progress reports for the srun command.
srun: job 11499365 has been allocated resources      |
[kejinhu@c0173 humanGenomeIndex]$ wget http://ftp.ensembl.org/pub/release-105/gtf/homo_sapiens/Homo_sapiens.GRCh38.105.gtf.gz # Download the human
--2021-12-19 13:01:57--  http://ftp.ensembl.org/pub/release-105/gtf/homo_sapiens/Homo_sapiens.GRCh38.105.gtf.gz                 GTF file from ENSEMBL.
Resolving ftp.ensembl.org (ftp.ensembl.org)... 193.62.197.76
Connecting to ftp.ensembl.org (ftp.ensembl.org)|193.62.197.76|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 50829040 (48M) [application/x-gzip]
Saving to: 'Homo_sapiens.GRCh38.105.gtf.gz'

100%[=====================================================================================================>] 50,829,040   476KB/s   in 1m 55s

2021-12-19 13:03:52 (431 KB/s) - 'Homo_sapiens.GRCh38.105.gtf.gz' saved [50829040/50829040]
                                                                                                                          # Time spent
[kejinhu@c0173 humanGenomeIndex]$ ls # List the downloaded file.                                                         for downloading
Homo_sapiens.GRCh38.105.gtf.gz
[kejinhu@c0173 humanGenomeIndex]$ ls -hl # List the downloaded file and its size.
total 97M
-rw-rw-r-- 1 kejinhu kejinhu 49M Jun 18  2021 Homo_sapiens.GRCh38.105.gtf.gz
[kejinhu@c0173 humanGenomeIndex]$ gunzip Homo_sapiens.GRCh38.105.gtf.gz # Uncompress the .gz file using the command of gunzip.
[kejinhu@c0173 humanGenomeIndex]$ ls -hl# List the uncompressed file and its size using ls with the options of -l and -h (combined as -hl).
total 2.6G
-rw-rw-r-- 1 kejinhu kejinhu 1.3G Jun 18  2021 Homo_sapiens.GRCh38.105.gtf# The file size becomes much greater after uncompression, and the .gz
[kejinhu@c0173 humanGenomeIndex]$                                                                                   file is gone.
```

**Figure 21**. Download the human GTF file to HPC and unzip it. Underlined in red is the login node; in yellow is the compute node. Texts after # are illustration of the scripts, their status, or outputs files.

After unzip the GTF file, you can see the content of this GTF file with the less command,

```
less Homo_sapiens.GRCh38.105.gtf
```

But the content is kind of messy when you check with the less command. You can see the more organized contents of the GTF file when you open it in the nano text editor,

nano Homo_sapiens.GRCh38.105.gtf    # You likely find that your job is killed when you try to open the GTF file using nano. This is because this file is 1.3 G and your requested memory is only 1 G using the default options of srun. To solve this problem, you can request more resources. First, you cancel the current interactive job using scancel [job ID]. Then request more resources, for example, srun --mem=3G --pty /bin/bash. Now, you are able to open the GTF file using nano. Be patient when you will see a black screen. It may take a while to open this file with nano because of its huge size.

## 4.2 Download and unzip the human genome FASTA file

You also need the human genome FASTA files to prepare the human genome indices for subsequent alignment. We use the wget command to download it as well. First, we request compute resources using the srun command,

```
srun --nodes=1 --cpus-per-task=10 --mem-per-cpu=5G --ntasks=1 --pty /bin/bash
```

# To demonstrate how to use the options of the srun command, here we request unnecessarily more resources than downloading the GTF file above originally considering that FASTA file of human genome is much greater than its GTF file. With the requested resources here on Cheaha, it took around 23 minutes 8 seconds (on the Cheaha node c0173) to download the human genome FASTA file using the code below. However, when the default resource (1 CPU, 1 G of memory using the script of srun --pty bash) was requested it spent even less time to download the same FASTA file of human genome (11 minutes 26 seconds with node c0170; or 11 minutes 36 seconds on node c0173, upward arrow in Figure 22).

Then we use the wget command to download the FASTA file of human genome. On any web browser, go to the ENSEMBL genome browser (https://useast.ensembl.org/index.html). On the ENSEMBL site: choose the human genome, and under the Genome Assembly pane, click the HTML text "*download DNA sequence (FASTA)*" (highlighted and annotated in the left pane of Figure 19), or you can click on the download thumbnail next to its HTML text; (Alternatively, in the Gene Annotation pane, click "*download FASTA*", and in the new windows, click the HTML text *dna/*, and this will take you to the same page). You will see a long list of FASTA files of the human genome, including files for each chromosome and the entire genome. You also have files with the repeat sequences masked by replacing the repeat sequences with Ns (rm files, RepeatMasked) or by marking the repeat sequences by lowercase letters (soft masked, sm). There are also files of "primary assembly" and "toplevel". Right-click on the HTML text for the soft-masked primary assembly and choose the clickable "*Copy Link*" text on the popup menu. Go back to the HPC terminal, and paste the link after the wget command,

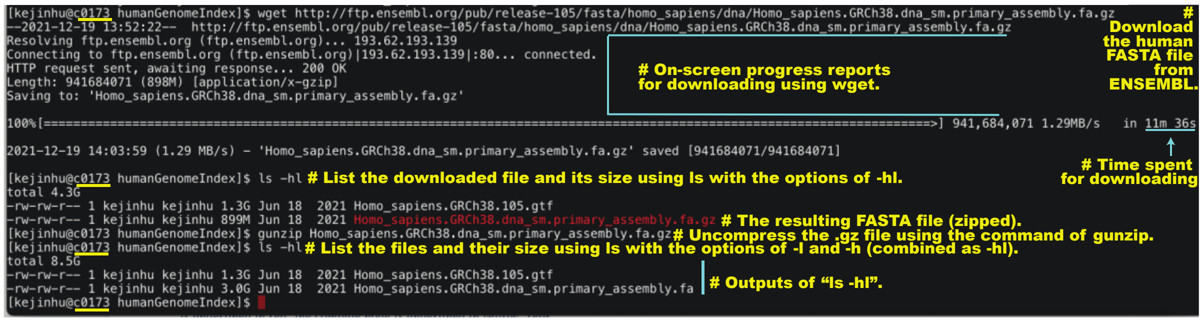
```
wget http://ftp.ensembl.org/pub/release-105/fasta/homo_sapiens/dna/Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa.gz
```

It is recommended that the human genome FASTA files are downloaded from the ENSEMBL database rather than NCBI or GENCODE. Linux is case sensitive, and you need to pay attention to the capital letters in the path. "sm" in the file name means soft masked, i.e., the repetitive sequences such as Alu and LINE are in lower case. It is recommended that "rm", which means "repeat masked", files should not be used because it masks the repetitive sequences by replacing them with Ns. It is also recommended that "primary_assembly" files are used instead of the "toplevel" files. Soft masked version should be used for STAR since STAR allows alignment to the marked regions while STAR can still detect which regions are masked.

After downloading, you will see the compressed *.gz* file in the humanGenomeIndex directory using the ls command with the -hl options. You will see a file of Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa.gz, which is only around 899 MB for the 105 version (Figure 22). To unzip the *gz* file, run the following code,

```
gunzip Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa.gz
```

# Human genome is huge, but the unzip process took several seconds only on Cheaha. To see the unzipped files, use the ls command (you can use the long option of -l with ls to see the sizes of the files. The combined -lh option will give the file sizes in the format more understandable, i.e., human readable.) (Figure 22). You should see a file of Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa. This file is 3 G in size. Please note that after unzipping, the zipped file is removed automatically.



```
[kejinhu@c0173 humanGenomeIndex]$ wget http://ftp.ensembl.org/pub/release-105/fasta/homo_sapiens/dna/Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa.gz
--2021-12-19 13:52:22--  http://ftp.ensembl.org/pub/release-105/fasta/homo_sapiens/dna/Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa.gz
Resolving ftp.ensembl.org (ftp.ensembl.org)... 193.62.193.139
Connecting to ftp.ensembl.org (ftp.ensembl.org)|193.62.193.139|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 941684071 (898M) [application/x-gzip]
Saving to: 'Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa.gz'

100%[==========================================================================================================>] 941,684,071 1.29MB/s   in 11m 36s

2021-12-19 14:03:59 (1.29 MB/s) - 'Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa.gz' saved [941684071/941684071]

[kejinhu@c0173 humanGenomeIndex]$ ls -hl # List the downloaded file and its size using ls with the options of -hl.
total 4.3G
-rw-rw-r-- 1 kejinhu kejinhu 1.3G Jun 18  2021 Homo_sapiens.GRCh38.105.gtf
-rw-rw-r-- 1 kejinhu kejinhu 899M Jun 18  2021 Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa.gz # The resulting FASTA file (zipped).
[kejinhu@c0173 humanGenomeIndex]$ gunzip Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa.gz# Uncompress the .gz file using the command of gunzip.
[kejinhu@c0173 humanGenomeIndex]$ ls -hl# List the files and their size using ls with the options of -l and -h (combined as -hl).
total 8.5G
-rw-rw-r-- 1 kejinhu kejinhu 1.3G Jun 18  2021 Homo_sapiens.GRCh38.105.gtf
-rw-rw-r-- 1 kejinhu kejinhu 3.0G Jun 18  2021 Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa
[kejinhu@c0173 humanGenomeIndex]$
```

# Download the human FASTA file from ENSEMBL.
# On-screen progress reports for downloading using wget.
# Time spent for downloading
# Outputs of "ls -hl".

**Figure 22**. Download the human genome FASTA file using the wget command, and uncompress it using the gunzip command. The compute node is underlined in yellow. In this example, both downloading (GTF and FASTA files) were completed on the same resource using the default options of srun since both tasks are light. Yellow texts after # are illustration of the Linux processes in screenshot.

After you unzip the FASTA files, you can open it in the nano facility and review the FASTA file. Or, you can check it using the less command,

```
nano Homo_sapiens.GRCh38.dna.primary_assembly.fa
```

Or,

```
less Homo_sapiens.GRCh38.dna.primary_assembly.fa
```

## 4.3 Generate human genome indices with the default threading parameter

We will submit the indexing script using the sbatch command. First, we make a SLURM file of STAR_Index.slurm using the nano text editor.

```
nano STAR_Index.slurm          # This command opens the nano text editor with the file name of STAR_Index.slurm even though you do not have such a file name beforehand.
```

In the nano text editor, prepare the following bash script to be submitted by the sbatch SLURM command,

```
#!/bin/bash
# STAR human genome indexing
#SBATCH --job-name=genomeIndexing
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --time=12:00:00
#SBATCH --mem=34G
#SBATCH --mail-type=FAIL
#SBATCH --mail-user=hukejin@gmail.com
#SBATCH --error=Index.err
#SBATCH --out=index.out
#SBATCH --chdir=/data/user/kejinhu/RNA_seq_tutorial/\
humanGenomeIndex/stdout_stderr/

module load STAR

STAR --runMode genomeGenerate \
--genomeDir /data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/ \
--genomeFastaFiles /data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/\
Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa \
--sjdbGTFfile /data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/\
Homo_sapiens.GRCh38.105.gtf \
--sjdbOverhang 50
```

Then, submit the job using the sbatch command,

```
sbatch STAR_Index.slurm
```

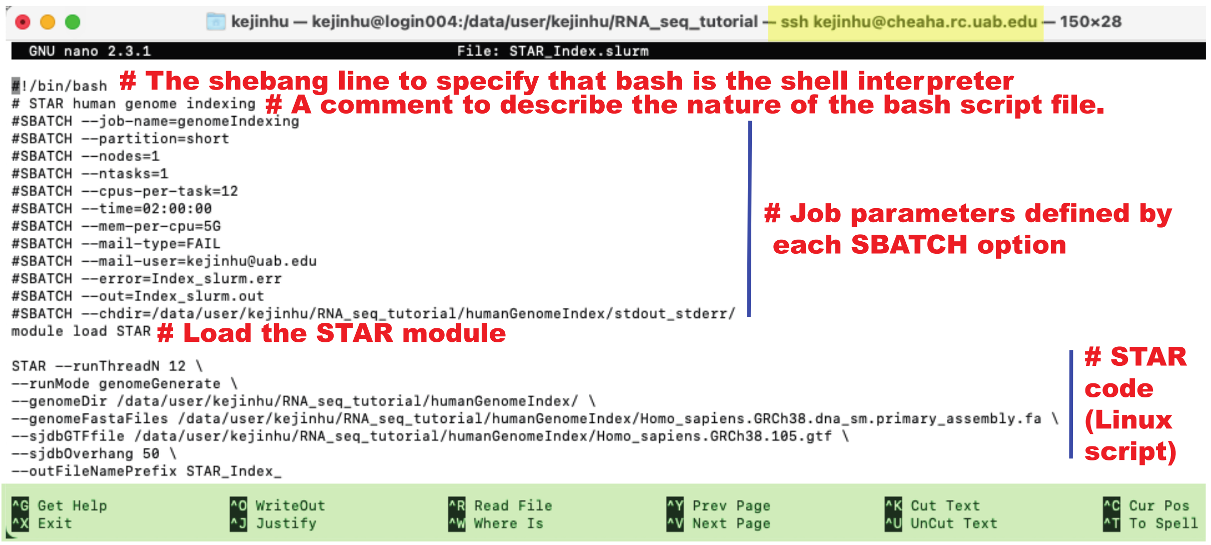
By default, STAR uses one CPU for indexing if not specified and therefore we specify --cpus-per-task = 1. The above code uses one CPU and 34 G of memory to index the human genome and spend 3 hours and 13 minutes to complete. Increasing the memory to 60 G did not improve the indexing speed and spent around 3 hours and 21 minutes to complete.

### 4.4 Generate human genome indices with multiple threading

STAR implements multiple threading, i.e., it uses multiple CPUs to conduct the same job. It can divide a job into multiple portions and each CPU will conduct a portion of the job. To this end, STAR has a --runThreadN option. The default value of the --runThreadN is 1. Indexing of human genome is very slow, but we can increase the number of CPUs to speed up the indexing. Prepare a file of STAR_indexN.slurm in nano text editor,

```
#!/bin/bash
# STAR human genome indexing
#SBATCH --job-name=genomeIndexing
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=12
#SBATCH --time=02:00:00
#SBATCH --mem-per-cpu=5G
#SBATCH --mail-type=FAIL
#SBATCH --mail-user=hukejin@gmail.com
#SBATCH --error=Index.err
#SBATCH --out=index.out
#SBATCH –chdir=/data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/stdout_stderr/

module load STAR

STAR --runThreadN 12 \
--runMode genomeGenerate \
--genomeDir /data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/ \
--genomeFastaFiles /data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/\
Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa \
--sjdbGTFfile /data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/Homo_\
sapiens.GRCh38.105.gtf \
--sjdbOverhang 50
```

The first line in the above code is called shebang line, which specifies that bash will be the shell interpreter. We specify this by providing the path to the bash software. Shebang consists of # and ! with no space between them. Usually we use #!/bin/bash without space between shebang and the interpreter, but spaces are allowed between #! and /bin/bash, and therefore #! /bin/bash will work. The second line is a comment describing the nature of the bash script after the SBATCH script. In Linux the pound sign # indicates a comment which is ignored by the bash interpreter. Each #SBATCH defines a sbatch command option. Please note that there is no space between # and SBATCH. If there is a space, it will be treated as a comment as in the case of line 2 in the above sbatch script. If the --out= option is not defined, the standard output file will be saved as a file name of slurm-[job_ID].out.



```
kejinhu — kejinhu@login004:/data/user/kejinhu/RNA_seq_tutorial — ssh kejinhu@cheaha.rc.uab.edu — 150×28
 GNU nano 2.3.1                          File: STAR_Index.slurm

#!/bin/bash # The shebang line to specify that bash is the shell interpreter
# STAR human genome indexing # A comment to describe the nature of the bash script file.
#SBATCH --job-name=genomeIndexing
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=12
#SBATCH --time=02:00:00
#SBATCH --mem-per-cpu=5G
#SBATCH --mail-type=FAIL
#SBATCH --mail-user=kejinhu@uab.edu
#SBATCH --error=Index_slurm.err
#SBATCH --out=Index_slurm.out
#SBATCH --chdir=/data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/stdout_stderr/
module load STAR # Load the STAR module

STAR --runThreadN 12 \
--runMode genomeGenerate \
--genomeDir /data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/ \
--genomeFastaFiles /data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa \
--sjdbGTFfile /data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/Homo_sapiens.GRCh38.105.gtf \
--sjdbOverhang 50 \
--outFileNamePrefix STAR_Index_

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

# Job parameters defined by each SBATCH option
# STAR code (Linux script)

**Figure 23**. Writing a sbatch script file for indexing human genome in the nano text editor. The nano text editor is a simple Linux text editor, and it even provides on-screen reference for shortcuts of some common functionalities at the bottom of your nano (shaded in green). In the shortcuts, ^ means the "Control" key. HPC login was achieved via SSH in this case (upper right, shaded in yellow). The SBATCH working directory is modified using the --chdir option. The red texts after the red # are illustration by the author.

After saving the STAR_Index.slurm, exit the nano text editor and submit the job from any directory as follows,

> sbatch /data/user/kejinhu/RNA_seq_tutorial/STAR_Index.slurm # This will submit the sbatch job with the script file of STAR_Index.slurm. The status of the submitted job can be checked using the command of squeue -u [your_user_name]. A PD status indicates that your job is pending, and an R status indicates your job is running. You can submit from any directory if you use the full path. You do not need to include the path if you submit it in the directory where the file STAR_Index.slurm is located. The STAR_Index.slurm file can be in any directory. The Log.out, slurm_jobID.out, and the Index.err files, and the _STARtmp directory will be saved in the same directory defined by the "change directory" --chdir= option of the SBATCH script. The resulting index files will be saved in the directory defined by the --genomeDir option of the STAR commend.

If you have no nano on your HPC, you may use the vi or vim text editor which is universal but needs more specialized skills to use. One practical difference between the nano text editor and the Word text editor is that nano has no automatic line wrapping. For convenience, the long STAR command line is broken into several lines in the above description. But the content of the long STAR command above (with many options, and long values for several options) should be in one line in the nano text editor when you do not introduce any operator within the command. Do not use Enter key to break the STAR command line into several lines when the STAR a command line become too long because this will change the meaning of the code. There is a Justify function in nano i.e., CTRL + J, but do not try to use it. Your code will not run if you break one command into 2 or several lines using CTRL+J. If you want the command line to continue in a different line, you can use the backslash operator \ to break the long command line into 2 or more lines as used in the above code. Please note that the backslash should be the very last character of your script line and a hidden mistake could be a space after the backslash. If there is a space after \, your job will abort. You will receive a failure email notice in this case. You can review the standard error (with the .err file extension) file and troubleshoot it. Used in this way, the backslash operator means continued in the next line (line continuation operator). Please note that a continuation operator \ is used for both file paths of the GTF and FASTA files to break each long command line (long file names and long directory names) into two lines. Unlike continuation operator used between two options in this code, there is no space before the continuation operator \ in these two file paths because the separated parts of each path should constitute one string.

The sbatch command is used to submit batch script to be run on your HPC. On the HPC terminal, issue sbatch --help to learn more about each option of the command sbatch such as --time, --error, --partition, and others. You can also use the manual command, man sbatch to get more detailed information about usage of the sbatch command. For the option of --mail-user, it is not necessary to use your work/institution email and your personal email will work. For the --partition option, please contact your HPC team (system administrator) to find out the available values defined for your HPC by your institution. You can find out the partition information by yourself using scontrol show partition as discussed previously. These are different from Cheaha depending on the capacity of your HPC. The --error option is useful in the case your codes have any bug. This option will generate a file containing the information about the failure. If the path for the --error option is not specified, the .err file will be saved where you submit the SBATCH file, but you can save it in any directory you define, e.g., --error=/path/to/the/desired/directory/index.err.

The working directory is the one the sbatch script file is submitted. Therefore, the standard output, standard error and the Log.out files as well as the _STARtmp directory are saved in the directory where you submit the SLURM file (working directory), not where the SLURM file is located. However, like the Log.out file, the standard output and standard error files can also be directed to any directory if the path is given. When the file name of the --error and --out are not defined, they will be generated automatically as slurm_%j.err and slurm_%j.out, where %j is the slurm job number. The working directory can be defined using the --chdir option. When the working directory is defined by the --chdir option, you can still save the stdout and stderr files in any directory you define individually in the --out and --error options. The directory of the Log.out file can be defined using the STAR option of --outFileNamePrefix (see below).

For usage of various options of the STAR command, visit https://github.com/alexdobin/STAR/blob/master/doc/STARmanual.pdf. You can also find online help,

```
module load STAR

STAR --help    # Please note that you need to load the software (STAR here) before
you could bring about its onscreen help information or build-in manual.
```

In the above code of genome indexing, we define the basic options of STAR command. For advanced options, refer to the STAR manual.

--runThreadN, defines the number of cores for indexing the genome. Here, we request 12 CPU cores.

--runMode, for indexing, the value for this option is genomeGenerate.

--genomeDir, define the path to the directory where genome indices will be saved upon generation.

--sjdbGTFfile, defines the path to the GTF file and the GTF file name. Please note that GTF file can be in a different directory from that defined by the --genomeDir option.

--genomeFastaFiles, defines the path to the FASTQ files, and the names of the FASTA files. Please note that FASTA files can be in a different directory from that defined by the --genomeDir option.

--sjdbOverhang, the value for this option is the ReadLength minus one. In this tutorial, all read length is 51 bases. Therefore, we use 51-1 = 50. RNA-seq read length is 100 for many experiments, and you should use 99 in that case. The STAR manual states that "the default value of 100 will work as well as the ideal value". Please note that the read length for --sjdbOverhang is the mate read length as reported in the FastQC results, not the average input read length as reported in the mapping/alignment output file Log.final.out, in which the average read length is 2× (mate read length) (see below). You can also easily find out the sequencing read length using IGV. In the alignment track of IGV, the read length is included in the popup text when you hover or click your mouse over the alignment track.

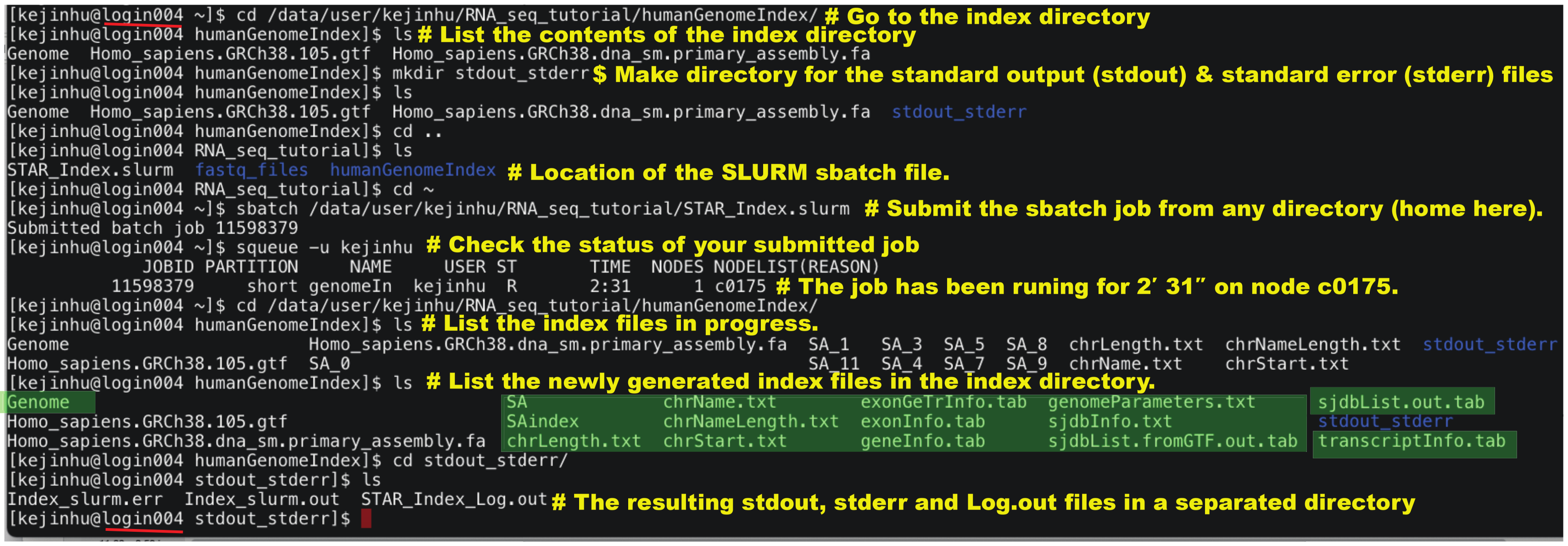
Using the above requested resources, Cheaha used 31 minutes to complete the indexing of human genome. Increasing the threading from 12 to 24 with the same memory (--mem=60G) shortens the operation time only to 20 minutes and 33 seconds. Indexing human genome requires large memory, but further increase of memory from 60 G to 120 G with the same number of CPUs (12 CPUs) showed no benefit and spent the same time to index human genome.

At this point, if you go to the directory of humanGenomeIndex you may see the file of Log.out (if the prefix is not defined using the --outFileNamePrefix option. When you defined the prepfix, the file name may be chosenName_Log.out). Your Log.out file may be in the directory where the script file is submitted if its destination is not defined. The Log.out file for STAR indexing can be placed in any directory if its path is defined using the --outFileNamePrefix option. This file is useful since it records the version of STAR and the codes (parameters) used for indexing (at the beginning of the file). You can use nano (command is nano Log.*out*) to open the Log.out file and check the basic information. You can also see the content of the Log.out file using: cat Log.out. Please note that in addition to the sequence of the 24 chromosomes, you can see many unlocalized genomic contigs. In the GRCh38.105 assembly, there are 169 contigs in addition to the chromosomes and mitochondria DNA. The full list of chromosomes and contigs is stored in the file of chrName.txt. Make sure you have those unlocalized genomic contig included. Otherwise, many reads will be reported as unmapped, and even worse mapped to the wrong places of the genome. You will have those unlocalized genomic contigs if you use the primary assembly of the FASTA files. You can see the lengths in bp for all chromosomes and contigs by opening the file of chrLength.txt.

In the directory of humanGenomeIndex, you can list all the files with ls -lh, and you will see the SA file is 23 G, and the Genome file is 3 G. Other files are much smaller.

In the genomeParameters.txt file, you can see the parameters you have used to generate the human genome index.

Please note that you do not need the files of Index.out, Index.err, and Log.out for the alignment using STAR. These files are saved in the directory where the SBATCH script file is submitted when their paths are not defined. The file of Log.out can be placed in any directory using the option of --outFileNamePrefix /path/to/the/desired/directory. The SLURM standard output and standard error file can be saved in any directory if specified in the #SBATCH --output= and #SBATCH --error= options. The genomeParameters.txt is needed for STAR alignment. This tutorial uses human genome as an example, and the audiences can generate genome indices for other species using the procedures here.



```
[kejinhu@login004 ~]$ cd /data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/ # Go to the index directory
[kejinhu@login004 humanGenomeIndex]$ ls # List the contents of the index directory
Genome  Homo_sapiens.GRCh38.105.gtf  Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa
[kejinhu@login004 humanGenomeIndex]$ mkdir stdout_stderr $ Make directory for the standard output (stdout) & standard error (stderr) files
[kejinhu@login004 humanGenomeIndex]$ ls
Genome  Homo_sapiens.GRCh38.105.gtf  Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa  stdout_stderr
[kejinhu@login004 humanGenomeIndex]$ cd ..
[kejinhu@login004 RNA_seq_tutorial]$ ls
STAR_Index.slurm  fastq_files  humanGenomeIndex # Location of the SLURM sbatch file.
[kejinhu@login004 RNA_seq_tutorial]$ cd ~
[kejinhu@login004 ~]$ sbatch /data/user/kejinhu/RNA_seq_tutorial/STAR_Index.slurm  # Submit the sbatch job from any directory (home here).
Submitted batch job 11598379
[kejinhu@login004 ~]$ squeue -u kejinhu  # Check the status of your submitted job
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          11598379     short genomeIn  kejinhu  R       2:31      1 c0175 # The job has been runing for 2′ 31″ on node c0175.
[kejinhu@login004 ~]$ cd /data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/
[kejinhu@login004 humanGenomeIndex]$ ls # List the index files in progress.
Genome                       Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa  SA_1   SA_3  SA_5  SA_8  chrLength.txt  chrNameLength.txt  stdout_stderr
Homo_sapiens.GRCh38.105.gtf  SA_0                                            SA_11  SA_4  SA_7  SA_9  chrName.txt    chrStart.txt
[kejinhu@login004 humanGenomeIndex]$ ls  # List the newly generated index files in the index directory.
Genome                                          SA             chrName.txt        exonGeTrInfo.tab  genomeParameters.txt      sjdbList.out.tab
Homo_sapiens.GRCh38.105.gtf                     SAindex        chrNameLength.txt  exonInfo.tab      sjdbInfo.txt              stdout_stderr
Homo_sapiens.GRCh38.dna_sm.primary_assembly.fa  chrLength.txt  chrStart.txt       geneInfo.tab      sjdbList.fromGTF.out.tab  transcriptInfo.tab
[kejinhu@login004 humanGenomeIndex]$ cd stdout_stderr/
[kejinhu@login004 stdout_stderr]$ ls
Index_slurm.err  Index_slurm.out  STAR_Index_Log.out # The resulting stdout, stderr and Log.out files in a separated directory
[kejinhu@login004 stdout_stderr]$
```

**Figure 24**. Snapshot of the indexing process. Shaded in green are the resulting files for indexing. Yellow texts after # are illustration. All the commands in this screenshot including submission of the job are light work and were conducted on the login node.

## Summary of Chapter 4

- The reference genome FASTA and annotation GTF files can be downloaded using the wget command.

- We use STAR with the genomeGenerate --runMode to produce the reference genome indices, which is required for the subsequent alignment of RNA-seq reads to the features of the reference genome.

- Human genome is huge, and it takes hours to index if the parameter is not optimized.

- STAR supports multiple threading, and the indexing speed can be increased significantly by using multiple CPUs via the --runThreadN option.

# Chapter 5

## Align the sequence reads to the reference genome

## 5.1 STAR Alignment of one sample and multiple threading of the STAR command

With the human genome indices established, we can now align the sequenced reads to the reference genome. This section introduces script for aligning a single RNA-seq sample to human genome using the STAR command with the *alignReads* runMode.

### 5.1.1 General script

In nano text editor, prepare the file of STAR_align_individual.slurm and save it in the fastq_files directory,

```
  #!/bin/bash
  # STAR alignment of an individual RNA-seq sample
  #SBATCH --job-name=STARalignment_individual
  #SBATCH --partition=express
  #SBATCH --nodes=1
  #SBATCH --ntasks=1
  #SBATCH --cpus-per-task=12
  #SBATCH --time=02:00:00
  #SBATCH --mem=40G
  #SBATCH --mail-type=FAIL
  #SBATCH --mail-user=kejinhu@uab.edu
  #SBATCH --error=align.err
  #SBATCH --out=align.out

  module load STAR

STAR --runMode alignReads --runThreadN 12 \
--genomeDir /data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex \
--outSAMtype BAM Unsorted \
--outFileNamePrefix SRR10203570 \
--readFilesIn  /data/user/kejinhu/RNA_seq_tutorial/fastq_files/SRR10203570_1.fastq  \
/data/user/kejinhu/RNA_seq_tutorial/fastq_files/SRR10203570_2.fastq
```

Then, submit this file in the directory of fastq_files,

```
sbatch STAR_align_individual.slurm
```

In the --readFilesIn option, the FASTQ files of the two sequencing mates should be separate by a space. For single-end RNA-seq data you just provide one file name as the value of the --readFilesIn. The path to the FASTQ files can be omitted here since the SLURM batch file is saved and will be submitted in the same directory of the FASTQ files, i.e., --readFilesIn SRR10203570_1.fastq SRR10203570_2.fastq.

All the operation/result files will be saved in the directory where the align_individual.slurm file is located and submitted. You can add a path to the --outFileNamePrefix option if you want to place alignment files somewhere else. You can save the stdout and stderr files in the directory you define for the --error, --out, or --chdir flags (with path values).

### 5.1.2 Increase alignment speed with multiple threading

Please note that STAR allows for multiple threading, which is parallel calculations at the data level (for the sample level parallelism of alignment, see Section 5.3). The above code uses 12 CPUs as defined in --cpu-per-task=12 and --runThreadN 12. You can find out the alignment speed of your script using the .progress.out files of STAR alignment results, for example,

```
cat SRR10203570_Log.progress.out
```

On Cheaha HPC, for counting a single RNA-seq sample the speed roughly doubles when the number of CPUs doubles before 8 CPUs, and the increasement of alignment speed slows down after that although it still increases till 12 CPUs, but the alignment speed reaches plateau beyond 16 CPUs. Here are the approximate alignment speeds for various number of CPUs on Cheaha for aligning the sample SRR1020370,

| | |
|---|---|
| 1 CPU | ~71 million reads/hour (M/h) |
| 2 CPUs | ~140 M/h |
| 4 CPUs | ~280 M/h |
| 8 CPUs | ~540 M/h |
| 10 CPUs | ~590 M/h |
| 12 CPUs | ~610 M/h |
| 16 CPUs | ~630 M/h |
| 24 CPUs | ~630 M/h |

### 5.1.3 Submit the alignment job as an interactive job

You can also directly run the above STAR code on bash terminal for alignment of a single sample without generating a file. To do this, you request resources first using srun, and run the above code on the pseudoterminal. For example, request resources using srun -c 12 -N 1 -n 1 --mem-per-cpu=3G --pty /bin/bash, and run the above STAR code directly as an interactive job. Please note that if you just request --mem=3G, you will see an error warning of “Bus error” after running the above STAR code because a total of 3G memory is too low for this work.

## 5.2 Serial alignment of multiple RNA-seq samples to the human genome

Submitting alignment job one sample each time is inefficient. We can submit one alignment job for all the RNA-seq samples of a project at one step using a bash for loop. After submission of the job the samples will be aligned one after one automatically (a serial alignment process). The alignment/mapping job need to be submitted via SLURM job scheduler as a non-interactive work since this process takes a lot of resources and time. We can make a directory of code_log where the SLURM code and SLURM stdout (standard output) and stderr (standard error) files will be saved when you will submit the job from this directory.

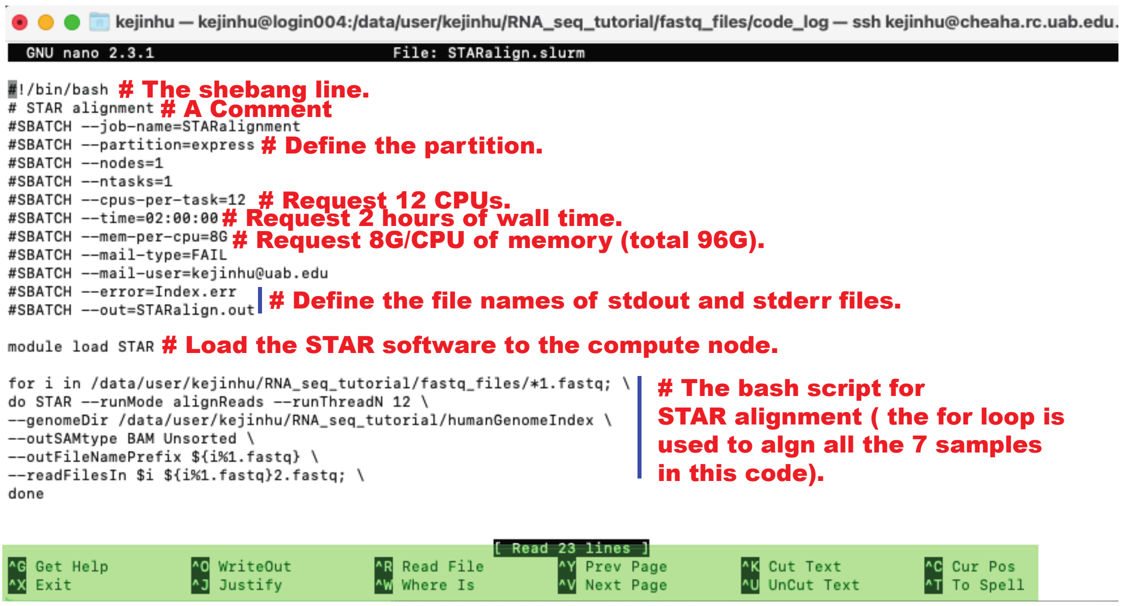
```
cd /data/user/kejinhu/RNA_seq_tutorial/fastq_files

mkdir code_log

nano STARserialAlign.slurm            # This opens the nano text editor with the
file name of STARserialAlign.slurm.
```

In nano, prepare a script file named STARserialAlign.slurm with the following code,

```
#!/bin/bash
# STAR serial alignment of multiple samples
#SBATCH --job-name=STARserialAlignment
#SBATCH --partition=express
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=12
#SBATCH --time=02:00:00
#SBATCH --mem-per-cpu=8G
#SBATCH --mail-type=FAIL
#SBATCH --mail-user=kejinhu@uab.edu
#SBATCH --error=serialAlign.err
#SBATCH --out=serialAlign.out

module load STAR

for i in /data/user/kejinhu/RNA_seq_tutorial/fastq_files/*1.fastq; \
do STAR --runMode alignReads --runThreadN 12 \
--genomeDir /data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex \
--outSAMtype BAM Unsorted \
--outFileNamePrefix ${i%1.fastq} \
--readFilesIn $i ${i%1.fastq}2.fastq; done
```

Then, submit the SLURM file from the directory of code_log,

> sbatch STARserialAlign.slurm   # This submits the BATCH script in the file of STARserialAlign.slurm to SLURM in the directory where the file STARserialAlign.slurm is located. The full or relative path should be provided if the job is submitted from any other directory. But, the stderr and stdout files will be saved in the submitting directory when their paths are not defined.



```
kejinhu — kejinhu@login004:/data/user/kejinhu/RNA_seq_tutorial/fastq_files/code_log — ssh kejinhu@cheaha.rc.uab.edu.
  GNU nano 2.3.1                    File: STARalign.slurm

#!/bin/bash # The shebang line.
# STAR alignment # A Comment
#SBATCH --job-name=STARalignment
#SBATCH --partition=express # Define the partition.
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=12 # Request 12 CPUs.
#SBATCH --time=02:00:00 # Request 2 hours of wall time.
#SBATCH --mem-per-cpu=8G # Request 8G/CPU of memory (total 96G).
#SBATCH --mail-type=FAIL
#SBATCH --mail-user=kejinhu@uab.edu
#SBATCH --error=Index.err    | # Define the file names of stdout and stderr files.
#SBATCH --out=STARalign.out  |

module load STAR # Load the STAR software to the compute node.

for i in /data/user/kejinhu/RNA_seq_tutorial/fastq_files/*1.fastq; \  | # The bash script for
do STAR --runMode alignReads --runThreadN 12 \                        | STAR alignment ( the for loop is
--genomeDir /data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex \    | used to algn all the 7 samples
--outSAMtype BAM Unsorted \                                           | in this code).
--outFileNamePrefix ${i%1.fastq} \
--readFilesIn $i ${i%1.fastq}2.fastq; \
done

                                    [ Read 23 lines ]
^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

**Figure 25**. Screenshot of the SLURM BATCH script in nano text editor. Shaded in green are references of shortcuts for some nano functionalities. Red texts after # and vertical bars are illustration.

For the SLURM part of the code, please refer to the descriptions in the genome indexing section (Section 4.4), and consult the online sbatch manual,

> man sbatch   # This brings about the online manual for the sbatch command of the SLURM controller. It is very long. You can directly go to the specific options used here by page down or up. Hit the key q to quit the help manual page. You can also get help information by issuing sbatch -h or sbatch --help.

This code uses the bash for loop, which has a syntax of "for [variable] in [character string]; *do* [application code]; done". In the above case, i is the variable. The character string provides the file names and their absolute path. When the script file STARserialAlign.slurm is saved and submitted in the directory where the FASTQ files are located the path to the FASTQ files can be omitted. In this case it can be shortened to "for i in *1.fastq" and other codes are the same. The wild card * here is used to represent the unique part of each FASTQ file name, i.e., sample names (* denotes any element in the string with undefined number of elements). The backslash \ indicates continuation of a command line. Please note that the continuation operator \ should occupy the last position of the line and there should not even be a space after it.

Please refer to the STAR vignette about each option used here. The default value for the --runMode option is *alignReads*, and therefore it can be omitted here. For the --readFilesIn option, this code defines a procedure for alignment of the paired-end RNA-seq reads, $i and ${i%1.fastq}2.fastq. The operator % here means the substring 1.fastq is deleted, and in this code the deleted 1.fastq is replaced with 2.fastq. Please note that the two file names for the paired-end reads should be separated by a space in the script.

If the FASTQ files are compressed, you do not need to decompress them, and we just add the option of --readFilesCommand zcat in the STAR command. The option of --outFileNamePrefix defines the unique part of the output file names, i.e., the sample names. Again, we exclude the 1.fastq part (called substring) in the output file names by string manipulation using the % operator, which means the substring after it will be deleted. The output files are automatically saved in the same directory defined by the character string for the FASTQ source files. The location of the human index files is defined by the --genomeDir option. The --outSAMtype option defines the format of the output alignment files. The above code gives two values to this option, *BAM* and *Unsorted*. Please note that STAR has an internal sorting function, and you can define it by giving the --outSAMtype an additional value of *SortedByCoordinate*. Usually, the sorting is conducted after alignment using the SAMtools software (see below). With the Cheaha resources requested above, it took around 30 minutes to align the 7 samples used in this tutorial. On average, each alignment took slightly over 4 minutes, but the time for each sample may varies. Alignment is a slow process. This quick alignment is achieved because STAR is an ultrafast aligner with a multiple threading function. To achieve high speed alignment by multiple threads/CPUs, here we give a value of 12 to the option of --runThreadN. This means you are using 12 CPUs to align the same sample. The mapping speed is recorded in the file of .progress.out and you can find it out using,

```
cat SRR10203569_Log.progress.out
```

## 5.3 Mapping multiple RNA-seq samples to genome by sample parallelism

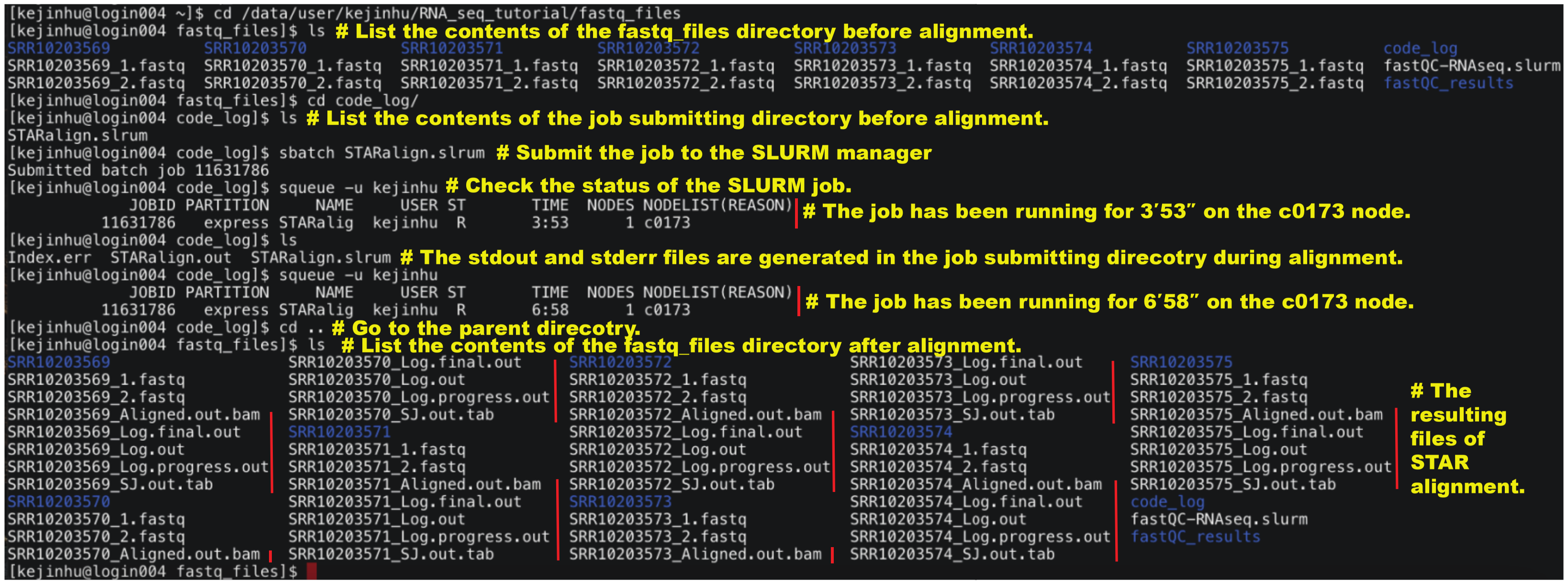
The above alignment code with a for loop map the RNA-seq samples one after one. The alignment time of an RNA-seq project can be further shortened using parallel calculation at the sample level. Parallel alignment allows mapping of many RNA-seq samples simultaneously. Here, I introduce sbatch array for parallelism. For an alternative method of parallel operation, see Section 7.3 (parallel counting of reads). In nano text editor, prepare the following code with a sbatch file name of parallel_align.slurm and save it in the cod_log directory,

```
#!/bin/bash
# Parallel STAR alignment at the sample level
```

```
#SBATCH --job-name=STARalignment_parallel
#SBATCH --partition=express
#SBATCH --nodes=1
#SBATCH --ntasks=7
#SBATCH --cpus-per-task=5
#SBATCH --time=02:00:00
#SBATCH --mem=100G
#SBATCH --mail-type=FAIL
#SBATCH --mail-user=hukejin@gmail.com
#SBATCH --error=align.err
#SBATCH --out=align.out
#SBATCH --array=0-6

module load STAR

READ1=("SRR10203569_1.fastq" "SRR10203570_1.fastq" "SRR10203571_1.fastq"
"SRR10203572_1.fastq" "SRR10203573_1.fastq" "SRR10203574_1.fastq"
"SRR10203575_1.fastq")
READ2=("SRR10203569_2.fastq" "SRR10203570_2.fastq" "SRR10203571_2.fastq"
"SRR10203572_2.fastq" "SRR10203573_2.fastq" "SRR10203574_2.fastq"
"SRR10203575_2.fastq")

STAR --runMode alignReads --runThreadN 5 \
--genomeDir /data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex \
--outSAMtype BAM Unsorted \
--outFileNamePrefix ../${READ1[$SLURM_ARRAY_TASK_ID]%_1.fastq} \
--readFilesIn ../${READ1[$SLURM_ARRAY_TASK_ID]} \
../${READ2[$SLURM_ARRAY_TASK_ID]}
```

In the above codes, we use two bash variables to define the sample names, *READ1* and *READ2*, each for one sequencing mate of a sample (paired-end sequencing). When defining a bash variable do not add any space around "="; you separate elements/strings by a space but not comma nor others (see code above).

We use the --array= flag with values of 0-6 (or 0, 1-6) to define 7 parallel tasks to be run simultaneously. Please note that the element positions for the bash array start at 0, not one, and therefore the array values are from 0 to 6 (not 1 to 7). The numbers of the variable SLURM_ARRAY_TASK_ID correspond to the number of the --array flag, i.e., 0-6, which define the elements in the *READ1* and *READ2* bash array variables by positions. Please note that you use the same values for the --cpus-per-task and --runThreadN flags here. The ../ in the --outFileNamePrefix and --readFilesIn options means the alignment and related files will be saved in the parent directory (the fastq_files directory), and that read the FASTQ files from the parent directory.



```
[kejinhu@login004 ~]$ cd /data/user/kejinhu/RNA_seq_tutorial/fastq_files
[kejinhu@login004 fastq_files]$ ls # List the contents of the fastq_files directory before alignment.
SRR10203569       SRR10203570       SRR10203571       SRR10203572       SRR10203573       SRR10203574       SRR10203575       code_log
SRR10203569_1.fastq  SRR10203570_1.fastq  SRR10203571_1.fastq  SRR10203572_1.fastq  SRR10203573_1.fastq  SRR10203574_1.fastq  SRR10203575_1.fastq  fastQC-RNAseq.slurm
SRR10203569_2.fastq  SRR10203570_2.fastq  SRR10203571_2.fastq  SRR10203572_2.fastq  SRR10203573_2.fastq  SRR10203574_2.fastq  SRR10203575_2.fastq  fastQC_results
[kejinhu@login004 fastq_files]$ cd code_log/
[kejinhu@login004 code_log]$ ls # List the contents of the job submitting directory before alignment.
STARalign.slrum
[kejinhu@login004 code_log]$ sbatch STARalign.slrum # Submit the job to the SLURM manager
Submitted batch job 11631786
[kejinhu@login004 code_log]$ squeue -u kejinhu # Check the status of the SLURM job.
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON) # The job has been running for 3′53″ on the c0173 node.
          11631786   express STARalig  kejinhu  R       3:53      1 c0173
[kejinhu@login004 code_log]$ ls
Index.err  STARalign.out  STARalign.slrum # The stdout and stderr files are generated in the job submitting direcotry during alignment.
[kejinhu@login004 code_log]$ squeue -u kejinhu
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON) # The job has been running for 6′58″ on the c0173 node.
          11631786   express STARalig  kejinhu  R       6:58      1 c0173
[kejinhu@login004 code_log]$ cd .. # Go to the parent direcotry.
[kejinhu@login004 fastq_files]$ ls  # List the contents of the fastq_files directory after alignment.
SRR10203569                  SRR10203570_Log.final.out     SRR10203572                  SRR10203573_Log.final.out     SRR10203575
SRR10203569_1.fastq          SRR10203570_Log.out           SRR10203572_1.fastq          SRR10203573_Log.out           SRR10203575_1.fastq
SRR10203569_2.fastq          SRR10203570_Log.progress.out  SRR10203572_2.fastq          SRR10203573_Log.progress.out  SRR10203575_2.fastq          # The
SRR10203569_Aligned.out.bam  SRR10203570_SJ.out.tab        SRR10203572_Aligned.out.bam  SRR10203573_SJ.out.tab        SRR10203575_Aligned.out.bam  resulting
SRR10203569_Log.final.out    SRR10203571                   SRR10203572_Log.final.out    SRR10203574                   SRR10203575_Log.final.out    files of
SRR10203569_Log.out          SRR10203571_1.fastq           SRR10203572_Log.out          SRR10203574_1.fastq           SRR10203575_Log.out          STAR
SRR10203569_Log.progress.out SRR10203571_2.fastq           SRR10203572_Log.progress.out SRR10203574_2.fastq           SRR10203575_Log.progress.out alignment.
SRR10203569_SJ.out.tab       SRR10203571_Aligned.out.bam   SRR10203572_SJ.out.tab       SRR10203574_Aligned.out.bam   SRR10203575_SJ.out.tab
SRR10203570                  SRR10203571_Log.final.out     SRR10203573                  SRR10203574_Log.final.out     code_log
SRR10203570_1.fastq          SRR10203571_Log.out           SRR10203573_1.fastq          SRR10203574_Log.out           fastQC-RNAseq.slurm
SRR10203570_2.fastq          SRR10203571_Log.progress.out  SRR10203573_2.fastq          SRR10203574_Log.progress.out  fastQC_results
SRR10203570_Aligned.out.bam  SRR10203571_SJ.out.tab        SRR10203573_Aligned.out.bam  SRR10203574_SJ.out.tab
[kejinhu@login004 fastq_files]$
```

**Figure 26**. Screenshot for the alignment process. Yellow texts are the author’s illustration.

Submit the sbatch file in the directory of code_log,

```
sbatch parallel_align.slurm
```

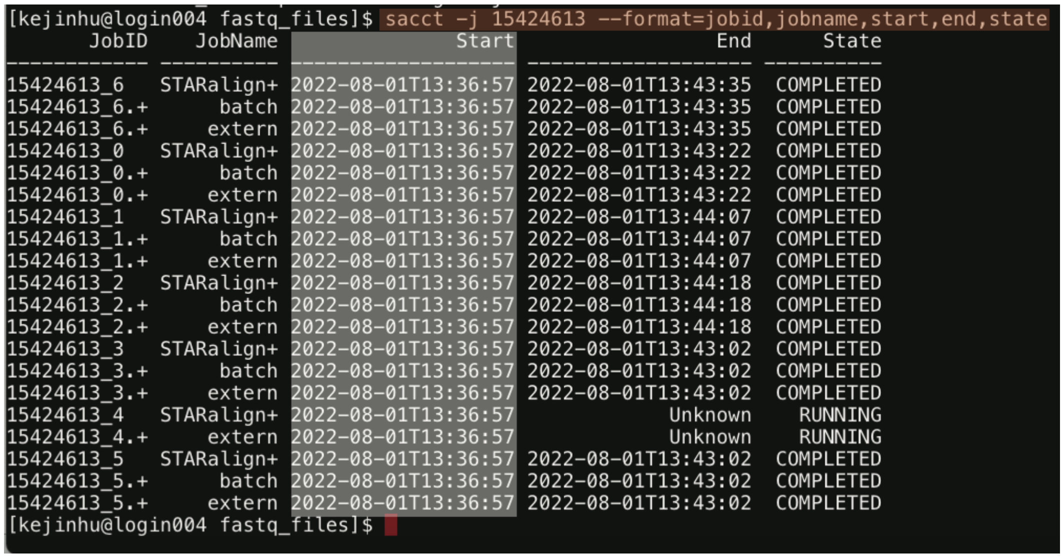
When the alignment is undergoing, you can check the job status using,

```
squeue -u [your_HPC_account_name]
```

From the above code, you can see the job/array ID, and you can further check the status of your array job using the SLURM command sacct (display job accounting data),

> sacct -j [array_ID] --format=jobid,jobname,start,end,state # The -j option means job or job step, and the job ID number is the value. Please note that there is no space anywhere (around = or between values) in the option of accounting format, i.e., "--format=jobid,jobname,start,end,state"(see Figure 27).

Please note that parallel alignment not necessarily shortens the alignment time. The above code started alignment of the 7 samples at the same time (shaded in Figure 27) and used less than 10 minutes to align the 7 samples here (compare the Start and End times for all the 7 tasks in Figure 27). However, if 2 CPUs ware used for each sample (the --runThreadN value is 2; parallelism at the data level), you do not see much improvement. This is because aligning individual samples become a bottleneck when less CPUs are used for aligning each sample. You can increase the thread number, but you likely wait for the resources from your HPC clusters, and as a result your job cannot be completed sooner because some samples may start late due to lack of compute resources from your HPC even though you use the array code. Therefore, we should balance between data and sample level parallelisms.



```
[kejinhu@login004 fastq_files]$ sacct -j 15424613 --format=jobid,jobname,start,end,state
       JobID    JobName               Start                 End      State
------------ ---------- ------------------- ------------------- ----------
15424613_6   STARalign+ 2022-08-01T13:36:57 2022-08-01T13:43:35  COMPLETED
15424613_6.+      batch 2022-08-01T13:36:57 2022-08-01T13:43:35  COMPLETED
15424613_6.+     extern 2022-08-01T13:36:57 2022-08-01T13:43:35  COMPLETED
15424613_0   STARalign+ 2022-08-01T13:36:57 2022-08-01T13:43:22  COMPLETED
15424613_0.+      batch 2022-08-01T13:36:57 2022-08-01T13:43:22  COMPLETED
15424613_0.+     extern 2022-08-01T13:36:57 2022-08-01T13:43:22  COMPLETED
15424613_1   STARalign+ 2022-08-01T13:36:57 2022-08-01T13:44:07  COMPLETED
15424613_1.+      batch 2022-08-01T13:36:57 2022-08-01T13:44:07  COMPLETED
15424613_1.+     extern 2022-08-01T13:36:57 2022-08-01T13:44:07  COMPLETED
15424613_2   STARalign+ 2022-08-01T13:36:57 2022-08-01T13:44:18  COMPLETED
15424613_2.+      batch 2022-08-01T13:36:57 2022-08-01T13:44:18  COMPLETED
15424613_2.+     extern 2022-08-01T13:36:57 2022-08-01T13:44:18  COMPLETED
15424613_3   STARalign+ 2022-08-01T13:36:57 2022-08-01T13:43:02  COMPLETED
15424613_3.+      batch 2022-08-01T13:36:57 2022-08-01T13:43:02  COMPLETED
15424613_3.+     extern 2022-08-01T13:36:57 2022-08-01T13:43:02  COMPLETED
15424613_4   STARalign+ 2022-08-01T13:36:57             Unknown    RUNNING
15424613_4.+     extern 2022-08-01T13:36:57             Unknown    RUNNING
15424613_5   STARalign+ 2022-08-01T13:36:57 2022-08-01T13:43:02  COMPLETED
15424613_5.+      batch 2022-08-01T13:36:57 2022-08-01T13:43:02  COMPLETED
15424613_5.+     extern 2022-08-01T13:36:57 2022-08-01T13:43:02  COMPLETED
[kejinhu@login004 fastq_files]$
```

**Figure 27.** Parallel alignments of 7 samples. Please note that all 7 tasks/alignments start at the same time (shaded with gray) but may end at different times. Here, task 4 is still running. The sacct command is shaded in red.

## 5.4 Briefly review the alignment output text files

STAR alignment generates 5 files for each sample in the fastq_files directory, _Aligned.out.bam, _Log.final.out, _Log.out, _Log.progress.out, and _SJ.out.tab. The .bam files are compressed files, and others are all text files. To find the type of file, you can call the command of file with the syntax of file your_file_name, for example,

```
file SRR10203575_Aligned.out.bam      # This returns file type information as "gzip
compressed data".

file SRR10203575_Log.out       # This reveals that it is an "ASCII text" file.
```

For the text files, you can briefly review them using the head, less, cat, or tail commands. For the short files (.Log.final.out), you can check with any command. For the long file, you can review using the less, head, or tail commands. To find out how many lines in each file, you can call the word count command, wc -l file_name. You can also find out the sizes of all files in the current directory by issuing ls -hl. As you can see that the BAM file is huge, and among the text files SJ.out.tab is the greatest one. Of course, for the text file you can examine using a text editor, for example the nano text editor. To open the text file in the nano text editor, just call, nano file_name, for example,

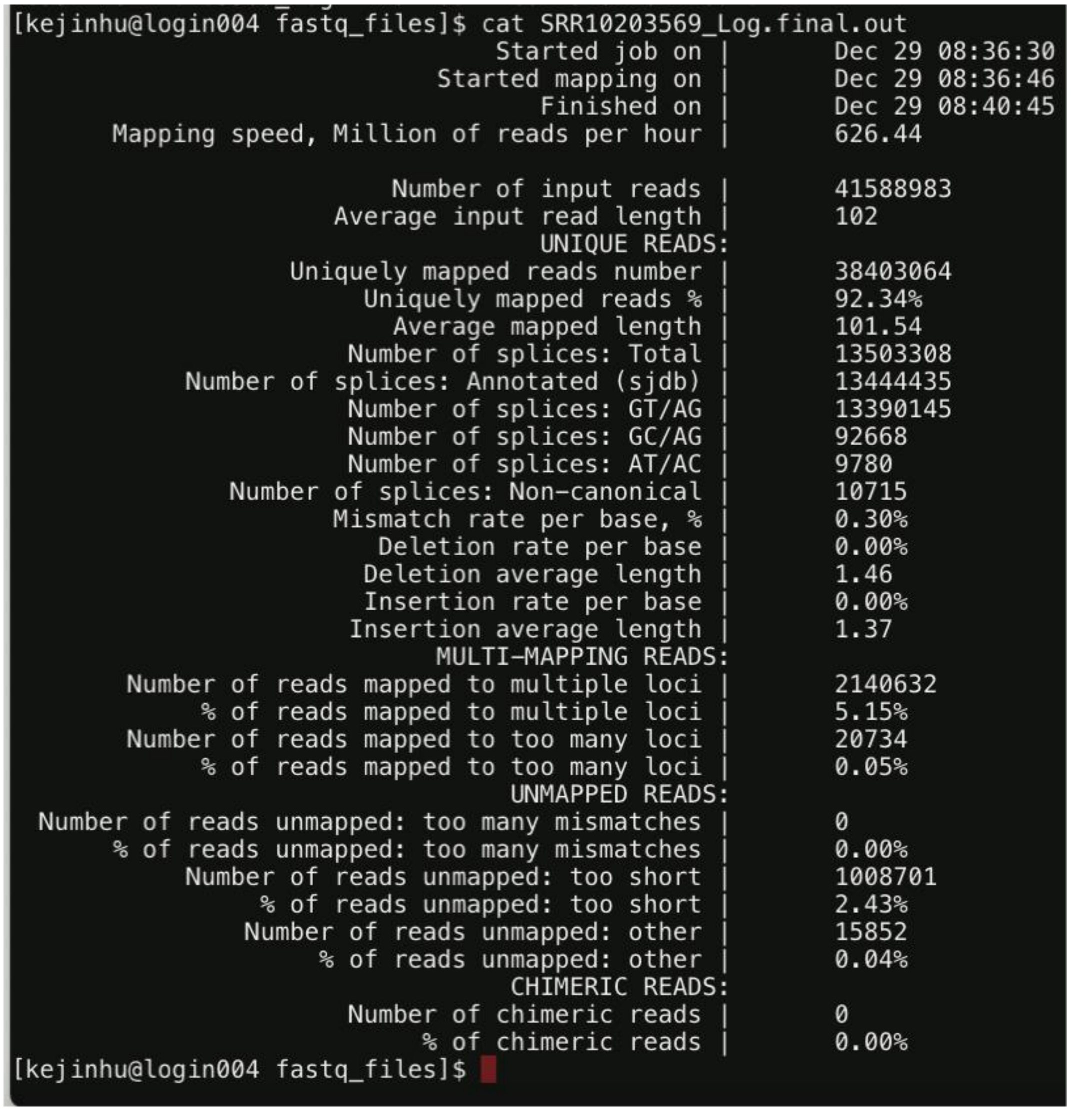
```
nano SRR10203575_Log.final.out       # This opens the text file in the nano text
editor.
```

The _Log.final.out files are useful since it contains statistics of the read counts. Let us output one of these files using the cat command,

```
cat SRR10203569_Log.final.out        # This outputs the mapping statistics (see
screenshot in Figure 28)
```

As you can see that the Log.final.out file provides many basic statistics for the alignment of read counts including input reads (total reads), number of the uniquely mapped reads, numbers of unmapped reads of different categories, statistics of reads with multiple matches, and others (Figure 28).

Of course, you can transfer these text files to your desktop using FileZilla or Globus, and then review them with your desktop text editor such as TextEdit. You may not be able to open the text files generated in Linux environment. This issue may be solved if you change the file extension to .txt from .out or .tab. With the .txt extension, these files can be opened using Excel.



```
[kejinhu@login004 fastq_files]$ cat SRR10203569_Log.final.out
                                 Started job on |       Dec 29 08:36:30
                             Started mapping on |       Dec 29 08:36:46
                                    Finished on |       Dec 29 08:40:45
       Mapping speed, Million of reads per hour |       626.44

                          Number of input reads |       41588983
                      Average input read length |       102
                                    UNIQUE READS:
                   Uniquely mapped reads number |       38403064
                        Uniquely mapped reads % |       92.34%
                          Average mapped length |       101.54
                       Number of splices: Total |       13503308
            Number of splices: Annotated (sjdb) |       13444435
                       Number of splices: GT/AG |       13390145
                       Number of splices: GC/AG |       92668
                       Number of splices: AT/AC |       9780
               Number of splices: Non-canonical |       10715
                      Mismatch rate per base, % |       0.30%
                         Deletion rate per base |       0.00%
                        Deletion average length |       1.46
                        Insertion rate per base |       0.00%
                       Insertion average length |       1.37
                             MULTI-MAPPING READS:
        Number of reads mapped to multiple loci |       2140632
             % of reads mapped to multiple loci |       5.15%
        Number of reads mapped to too many loci |       20734
             % of reads mapped to too many loci |       0.05%
                                  UNMAPPED READS:
  Number of reads unmapped: too many mismatches |       0
       % of reads unmapped: too many mismatches |       0.00%
            Number of reads unmapped: too short |       1008701
                 % of reads unmapped: too short |       2.43%
                Number of reads unmapped: other |       15852
                     % of reads unmapped: other |       0.04%
                                  CHIMERIC READS:
                       Number of chimeric reads |       0
                            % of chimeric reads |       0.00%
[kejinhu@login004 fastq_files]$
```

**Figure 28**. Screenshot for alignment statistics in the file of SRR10203569_Log.final.out.

## 5.5 Aggregate the Log.out files using MultiQC

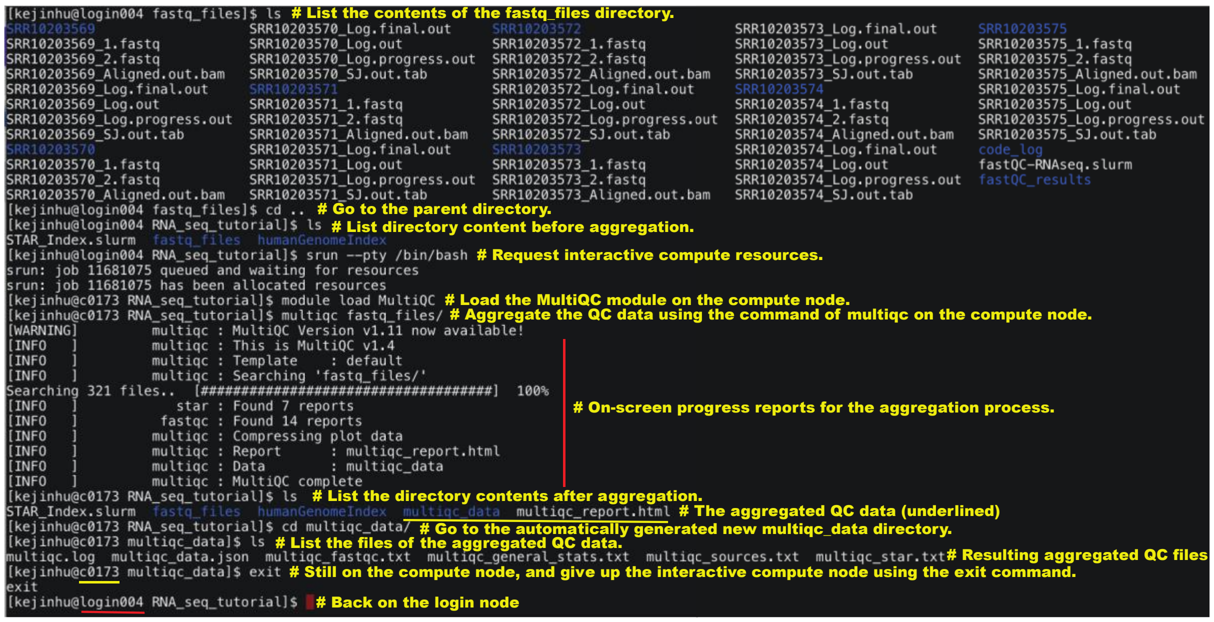
In the parent directory of the fastq_files, using the following code to aggregate the mapping summaries for all of the samples,

srun --pty bash                          # This requests compute resources using the default options. The multiqc operation is a light work but it still needs to be conducted on a compute node not the login node.

module load MultiQC

```
multiqc fastq_files
```

The above code generates one directory, multiqc_data and one HTML file in the parent directory of fastq_files. There are text files inside the multiqc_data directory, one log file (multiqc.log), one json code file (multiqc_data.jason), one source record file (multiqc_sources.txt), and three result summary files (multiqc_fastqc.txt, mutiqc_general_stats.txt, multiqc_star.txt). The multiqc_fastqc.txt also aggregates the FastQC results you generated previously since the results are inside the directory of fastq_files. The FastQC results are aggregated even though the results are placed in a separate directory under the directory of fastq_files. However, there is no multiqc_fastqc.txt file if you do not FastQC your FASTQ files first. In this case multiqc aggregates the STAR mapping results only, i.e., you have the multiqc.star.txt file only after you run the multiqc command on the fastq_files directory without any FastQC results. You can examine the results in text files with the nano text editor or the cat commands, or you can examine them with Excel or web browser after transferring them to your desktop computer.



```
[kejinhu@login004 fastq_files]$ ls  # List the contents of the fastq_files directory.
SRR10203569                 SRR10203570_Log.final.out     SRR10203572                 SRR10203573_Log.final.out     SRR10203575
SRR10203569_1.fastq         SRR10203570_Log.out           SRR10203572_1.fastq         SRR10203573_Log.out           SRR10203575_1.fastq
SRR10203569_2.fastq         SRR10203570_Log.progress.out  SRR10203572_2.fastq         SRR10203573_Log.progress.out  SRR10203575_2.fastq
SRR10203569_Aligned.out.bam SRR10203570_SJ.out.tab        SRR10203572_Aligned.out.bam SRR10203573_SJ.out.tab        SRR10203575_Aligned.out.bam
SRR10203569_Log.final.out   SRR10203571                   SRR10203572_Log.final.out   SRR10203574                   SRR10203575_Log.final.out
SRR10203569_Log.out         SRR10203571_1.fastq           SRR10203572_Log.out         SRR10203574_1.fastq           SRR10203575_Log.out
SRR10203569_Log.progress.out SRR10203571_2.fastq          SRR10203572_Log.progress.out SRR10203574_2.fastq          SRR10203575_Log.progress.out
SRR10203569_SJ.out.tab      SRR10203571_Aligned.out.bam   SRR10203572_SJ.out.tab      SRR10203574_Aligned.out.bam   SRR10203575_SJ.out.tab
SRR10203570                 SRR10203571_Log.final.out     SRR10203573                 SRR10203574_Log.final.out     code_log
SRR10203570_1.fastq         SRR10203571_Log.out           SRR10203573_1.fastq         SRR10203574_Log.out           fastQC-RNAseq.slurm
SRR10203570_2.fastq         SRR10203571_Log.progress.out  SRR10203573_2.fastq         SRR10203574_Log.progress.out  fastQC_results
SRR10203570_Aligned.out.bam SRR10203571_SJ.out.tab        SRR10203573_Aligned.out.bam SRR10203574_SJ.out.tab
[kejinhu@login004 fastq_files]$ cd ..  # Go to the parent directory.
[kejinhu@login004 RNA_seq_tutorial]$ ls  # List directory content before aggregation.
STAR_Index.slurm  fastq_files  humanGenomeIndex
[kejinhu@login004 RNA_seq_tutorial]$ srun --pty /bin/bash # Request interactive compute resources.
srun: job 11681075 queued and waiting for resources
srun: job 11681075 has been allocated resources
[kejinhu@c0173 RNA_seq_tutorial]$ module load MultiQC # Load the MultiQC module on the compute node.
[kejinhu@c0173 RNA_seq_tutorial]$ multiqc fastq_files/ # Aggregate the QC data using the command of multiqc on the compute node.
[WARNING]         multiqc : MultiQC Version v1.11 now available!
[INFO   ]         multiqc : This is MultiQC v1.4
[INFO   ]         multiqc : Template    : default
[INFO   ]         multiqc : Searching 'fastq_files/'
Searching 321 files..  [####################################]  100%
[INFO   ]            star : Found 7 reports                        # On-screen progress reports for the aggregation process.
[INFO   ]          fastqc : Found 14 reports
[INFO   ]         multiqc : Compressing plot data
[INFO   ]         multiqc : Report      : multiqc_report.html
[INFO   ]         multiqc : Data        : multiqc_data
[INFO   ]         multiqc : MultiQC complete
[kejinhu@c0173 RNA_seq_tutorial]$ ls  # List the directory contents after aggregation.
STAR_Index.slurm  fastq_files  humanGenomeIndex  multiqc_data  multiqc_report.html # The aggregated QC data (underlined)
[kejinhu@c0173 RNA_seq_tutorial]$ cd multiqc_data/ # Go to the automatically generated new multiqc_data directory.
[kejinhu@c0173 multiqc_data]$ ls # List the files of the aggregated QC data.
multiqc.log  multiqc_data.json  multiqc_fastqc.txt  multiqc_general_stats.txt  multiqc_sources.txt  multiqc_star.txt# Resulting aggregated QC files
[kejinhu@c0173 multiqc_data]$ exit # Still on the compute node, and give up the interactive compute node using the exit command.
exit
[kejinhu@login004 RNA_seq_tutorial]$ # Back on the login node
```

**Figure 29**. Aggregate the QC results of STAR alignment.

## 5.6 Explore the BAM files using SAMtools

The BAM files are binary form of SAM files, and only the latter are human readable. If You call head sample.bam, and it will display some messy code, or may not output anything at all. To see the meaningful BAM file, you should use the SAMtools utility. For formats of SAM/BAM files, you can refer to https://samtools.github.io/hts-specs/SAMtags.pdf [15].

module spider samtools # When you use module spider, the module names are not case sensitive. But, when you use module load, it is. You can even use the incomplete module name when you use the command of module spider or module avail, e.g., module spider samto, or module avail samt.

module load SAMtools # This code loads the latest version of SAMtools. You can also load a defined version by providing the full module name, for example, module load SAMtools/1.9-GCC-6.4.0-2.28. You will see a warning when you call "module load samtools"because it is case sensitive.

After loading the SAMtools, you can check its version and basic functionality. Try the following commands,

```
samtools --version

samtools

samtools --help

man samtools
```

Now, you can use samtools view to see the BAM files. Because a BAM file is huge, we just check the first a few rows using the pipe operator | and the head command. The pipe operator means using the standard output (stdout) of the first command (samtools view here) as the standard input (stdin) of the following command (head here),

samtools view my.bam | head # This prints out the data for the first 10 alignments of a BAM file of your interest. You can try any of the 7 BAM files in this tutorial. If you just want to check the first alignment, you can define it by giving a value to the -n option of the head command, samtools view my.bam | head -n 1. From the output of this command, we can see the reference sequence of each alignment, which is the 10th column of the SAM file. This is another way you can tell what is the read length of your RNA-seq protocol. Count the length of any of the aligned reference sequence and it is the ReadLength value to calculate the value for the option --sjdbOverhang when you build the genome indices described in Chapter 4.

The above code just gives you some of the alignments without the header information, and you can use the header option -h to include the header information,

samtools view -h my.bam | head # This displays the first 10 lines of the header information. You cannot see any alignment because the header information has almost 200 lines. Each chromosome or a contig occupies one line. At the end of the header, the STAR alignment code is included. You can see the STAR codes if you increase number of lines for the line option -n of the head command as, samtools view -h my.bam | head -n 200. The BAM file is large, and it takes much longer time to output the results if you use the tail command than the head command.

I you want to check more of the alignment BAM/SAM files including the header, you can include the less command along with the pipe operator |,

samtools view -h my.bam | less            # This allows you to examine your BAM/SAM file page by page, or line by line.

## Summary of Chapter 5

- We use the alignReads --runMode of the STAR command to align RNA-seq reads to features of the reference genome.

- Alignment of one individual RNA-seq sample can be sped up by multiple threading option of the STAR command (parallelism at the data level). Multiple threading can double the alignment speed when the number of CPUs double till 8 CPUs, and the increase of CPUs can further increase the alignment speed until you have around 12 CPUs.

- A code for serial alignment of multiple samples can save you the hands-on time.

- The alignment speed can be further increased by simultaneous alignment of multiple samples using a job array script (parallelism at the sample level).

# Chapter 6

## Sort, index and visually inspect the aligned BAM files

Before we count the reads to each feature and conduct the statistics analyses, we need to sort and index the BAM files.

## 6.1 The samtools sort syntax and sorting one sample

To demonstrate the syntax of the samtolls sort command, we first test sorting one BAM file first in the interactive mode using the code,

```
srun --pty bash

samtools sort -o my.sortedByCoord.bam -O bam my.bam     # The -o option means writing the final output to the file with a name defined in this option rather than as a standard output, while the -O option defines the output file format (BAM, SAM, or CRAM). Please note that by default samtools sort by coordination of chromosome positions, and therefore we add an informative tag of sortedByCoord to each sorted BAM file name.
```

## 6.2 Serial sorting and multiple threading at the data level

When sorting all the BAM files using one script we prepare a SLURM batch script file named samtools_sort.slurm using the for loop in nano text editor,

```
#!/bin/bash
# Sorting the BAM files using samtools sort
#SBATCH --job-name=SAMtoolsSort
#SBATCH --partition=short
#SBATCH --nodes=1
#SBATCH --cpus-per-task=1
#SBATCH --ntasks=1
#SBATCH --mail-type=FAIL
#SBATCH --mail-user=your.email@domain
#SBATCH --error=sort.err
#SBATCH --time=05:00:00
#SBATCH --mem-per-cpu=5G
#SBATCH --out=sort.out

module load SAMtools

for i in /data/user/kejinhu/RNA_seq_tutorial/fastq_files/*.bam; \
do samtools sort -o ${i%out.bam}sortedByCoord.bam -O bam $i; \
done
```

In the batch job code, you should define the path to your source BAM files. The sorted output files will be saved in the same directory of the source BAM files using the above code. However, the submitting directory of the script file samtools_sort.slurm is the working directory. You may save and submit the script file in the directory containing the target BAM files and make your code shorter (just "for i in *.bam" without the path). After preparing the code above in nano, submit the sorting job in the directory containing the script file using the sbatch command,

```
sbatch samtools_sort.slurm      # This establishes a non-interactive job for sorting
the BAM files.
```

In the BAM source file directory (fastq_files directory in this tutorial), you can see the resulting sorted files, and the temporary files of the sorted parts before merging. After the sorting is completed, you can see the list of the files with their sizes using ls -hl. You will see that the sorted BAM file is much smaller than its parent BAM file.

With the requested resources of Cheaha in the above code, it took around 103 minutes to sort the 7 BAM files in this tutorial. Increasing the number of CPUs from one to 12 did not improve sorting speed and need the same amount of time to sort the 7 samples. This is because by default samtools sort uses one CPU only. However, samtools sort does allow multiple threading (multiple CPUs), but you need to define it using the -@ or --thread option. When -@ 3, that means 3 CPUs are allowed, is defined it took around 39 minutes to sort the 7 samples. If the --thread value is further increased to 12, Cheaha took around 10 minutes to sort the 7 samples. However, further increase of CPUs from 12 to 24 for multiple threading improved the speed very little and Cheaha spent around 8 minutes and 20 seconds to complete sorting the 7 samples.

## 6.3 Combine parallel sorting and multiple threading

We can shorten the alignment time by parallel sorting at the sample level using the --array tag. Prepare the following sbatch file in nano text editor with a file name of parallel_sort.slurm,

```
#!/bin/bash
# Parallel sorting the BAM files using samtools sort

#SBATCH --job-name=SAMtoolsSort
#SBATCH --partition=express
#SBATCH --nodes=1
#SBATCH --ntasks=7
#SBATCH --cpus-per-task=1
#SBATCH --mail-type=FAIL
#SBATCH --mail-user=kejinhu@uab.edu
#SBATCH --error=sort.err
```

```
#SBATCH --time=02:00:00
#SBATCH --mem=50G
#SBATCH --out=sort.out
#SBATCH --array=0-6

module load SAMtools

FILE=("SRR10203569Aligned.out.bam"                "SRR10203570Aligned.out.bam"
"SRR10203571Aligned.out.bam"                      "SRR10203572Aligned.out.bam"
"SRR10203573Aligned.out.bam"                      "SRR10203574Aligned.out.bam"
"SRR10203575Aligned.out.bam")

samtools sort -o ${FILE[$SLURM_ARRAY_TASK_ID]%out.bam}sortedByCoord.bam -
O BAM ${FILE[$SLURM_ARRAY_TASK_ID]}
```

The above code uses a bash array variable FILE to define the list of BAM files to be sorted. Please note that there is not any space around the assignment sign "=" when defining an array variable. The 7 tasks are defined in the --array flag. Using 1 CPU per task, it took <17 minutes only to complete the soring of 7 samples (vs 105 minutes by serial sorting using more CPUs).

With the above code, increasing the number of CPUs from 1 to 2 or increasing the total memory from 50 G to 100 G, or increasing both all spent similar time to complete the sorting of the 7 samples. This is because samtools uses one CPU per task by default. SAMtools support multiple threading and conducts parallel sorting at the data level of a single file.  You can use the multiple threading option -@ to have multiple CPUs to sort each sample. In the above code, when you set -@ 2 (i.e., sort each sample using 2 CPUs), the total time for sorting the 7 samples was around 8 minutes (vs 17 minutes). Further increase of threads to 3 additionally improved the speed a little bit (around 5 minutes for sorting the 7 samples).

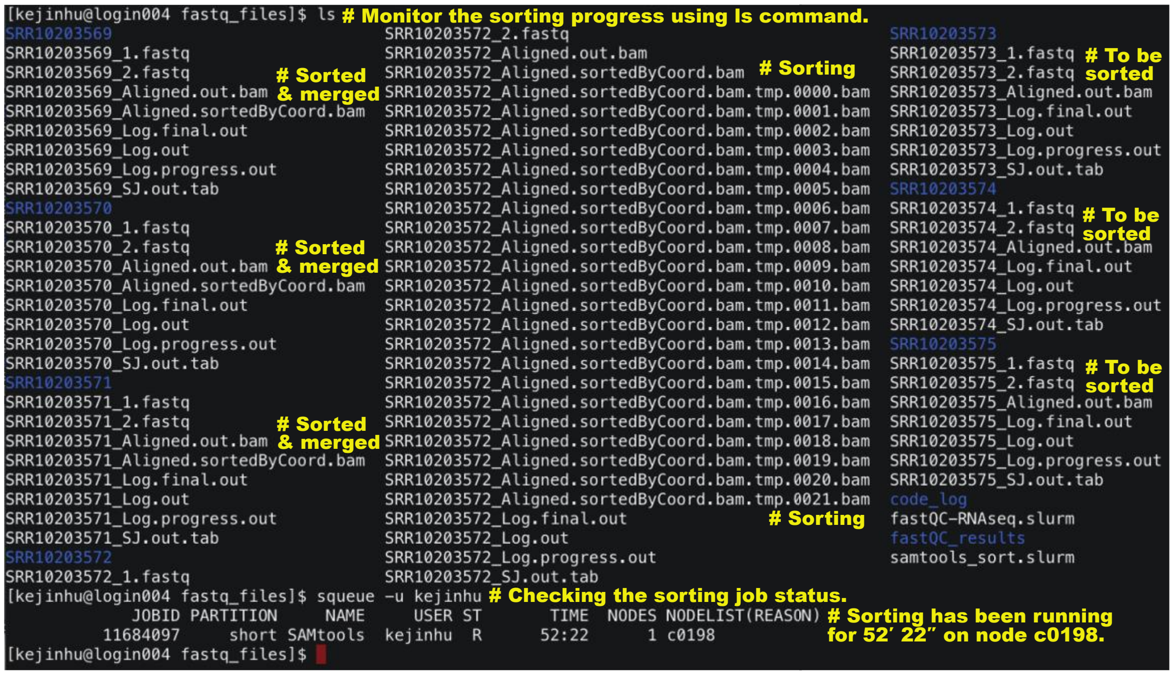
## 6.4 Index the sorted BAM files using SAMtools

The next step is indexing the sorted BAM files using the samtools index command. Indexing the sorted BAM files is a light work taking around 1 minute per sample, and we can use srun to submit it as an interactive job. Do not use the login nodes even for a work of 1 minute. It is said you can run a code/command on the login nodes only if it takes less than 1 second. You can index these files using the following srun SLURM code under the directory where your sortedByCoord.bam files are located.

```
cd /data/user/kejinhu/RNA_seq_tutorial/fastq_files
```

srun -c 1 -N 1 -n 1 --pty /bin/bash                # The option -N is equivalent to --nodes; option -n is the same as --ntasks; -c is the one-letter version of the long --cpus-per-task option.

samtools index -b my_sortedByCoord.bam        # This script automatically generates a file of my_sortedByCoord.bam.bai in the same directory. The -b option means to create an index file for the BAM file (BAM Index, BAI). The -b option is the default and therefore you can omit it, i.e., samtools index my_sortedByCoord.bam will work the same.



```
[kejinhu@login004 fastq_files]$ ls # Monitor the sorting progress using ls command.
SRR10203569                          SRR10203572_2.fastq                                SRR10203573
SRR10203569_1.fastq                  SRR10203572_Aligned.out.bam                        SRR10203573_1.fastq # To be
SRR10203569_2.fastq         # Sorted SRR10203572_Aligned.sortedByCoord.bam # Sorting    SRR10203573_2.fastq sorted
SRR10203569_Aligned.out.bam & merged SRR10203572_Aligned.sortedByCoord.bam.tmp.0000.bam SRR10203573_Aligned.out.bam
SRR10203569_Aligned.sortedByCoord.bam SRR10203572_Aligned.sortedByCoord.bam.tmp.0001.bam SRR10203573_Log.final.out
SRR10203569_Log.final.out            SRR10203572_Aligned.sortedByCoord.bam.tmp.0002.bam SRR10203573_Log.out
SRR10203569_Log.out                  SRR10203572_Aligned.sortedByCoord.bam.tmp.0003.bam SRR10203573_Log.progress.out
SRR10203569_Log.progress.out         SRR10203572_Aligned.sortedByCoord.bam.tmp.0004.bam SRR10203573_SJ.out.tab
SRR10203569_SJ.out.tab               SRR10203572_Aligned.sortedByCoord.bam.tmp.0005.bam SRR10203574
SRR10203570                          SRR10203572_Aligned.sortedByCoord.bam.tmp.0006.bam SRR10203574_1.fastq # To be
SRR10203570_1.fastq                  SRR10203572_Aligned.sortedByCoord.bam.tmp.0007.bam SRR10203574_2.fastq sorted
SRR10203570_2.fastq         # Sorted SRR10203572_Aligned.sortedByCoord.bam.tmp.0008.bam SRR10203574_Aligned.out.bam
SRR10203570_Aligned.out.bam & merged SRR10203572_Aligned.sortedByCoord.bam.tmp.0009.bam SRR10203574_Log.final.out
SRR10203570_Aligned.sortedByCoord.bam SRR10203572_Aligned.sortedByCoord.bam.tmp.0010.bam SRR10203574_Log.out
SRR10203570_Log.final.out            SRR10203572_Aligned.sortedByCoord.bam.tmp.0011.bam SRR10203574_Log.progress.out
SRR10203570_Log.out                  SRR10203572_Aligned.sortedByCoord.bam.tmp.0012.bam SRR10203574_SJ.out.tab
SRR10203570_Log.progress.out         SRR10203572_Aligned.sortedByCoord.bam.tmp.0013.bam SRR10203575
SRR10203570_SJ.out.tab               SRR10203572_Aligned.sortedByCoord.bam.tmp.0014.bam SRR10203575_1.fastq # To be
SRR10203571                          SRR10203572_Aligned.sortedByCoord.bam.tmp.0015.bam SRR10203575_2.fastq sorted
SRR10203571_1.fastq                  SRR10203572_Aligned.sortedByCoord.bam.tmp.0016.bam SRR10203575_Aligned.out.bam
SRR10203571_2.fastq         # Sorted SRR10203572_Aligned.sortedByCoord.bam.tmp.0017.bam SRR10203575_Log.final.out
SRR10203571_Aligned.out.bam & merged SRR10203572_Aligned.sortedByCoord.bam.tmp.0018.bam SRR10203575_Log.out
SRR10203571_Aligned.sortedByCoord.bam SRR10203572_Aligned.sortedByCoord.bam.tmp.0019.bam SRR10203575_Log.progress.out
SRR10203571_Log.final.out            SRR10203572_Aligned.sortedByCoord.bam.tmp.0020.bam SRR10203575_SJ.out.tab
SRR10203571_Log.out                  SRR10203572_Aligned.sortedByCoord.bam.tmp.0021.bam code_log
SRR10203571_Log.progress.out         SRR10203572_Log.final.out                # Sorting fastQC-RNAseq.slurm
SRR10203571_SJ.out.tab               SRR10203572_Log.out                                fastQC_results
SRR10203572                          SRR10203572_Log.progress.out                       samtools_sort.slurm
SRR10203572_1.fastq                  SRR10203572_SJ.out.tab
[kejinhu@login004 fastq_files]$ squeue -u kejinhu # Checking the sorting job status.
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON) # Sorting has been running
          11684097     short SAMtools  kejinhu  R      52:22      1 c0198            for 52′ 22″ on node c0198.
[kejinhu@login004 fastq_files]$
```

**Figure 30**. Screenshot for sorting the aligned BAM files. BAM Files of sorted, sorting, and to be sorted are labelled with yellow texts.

If you want to index all the BAM files in the working directory, use the for loop code below after you have been granted the resources requested by srun,

for i in *.sortedByCoord.out.bam; do samtools index -b $i $i.bai; done            # the $i.bai can be omitted in this code. The -b option is the default and therefore can be omitted as well.

When you index your sorted BAM files by submitting a batch SLURM file, you should specify the path to the source sorted BAM files, and the code is the same otherwise. You do not need to define the path if you save and submit the SLURM batch script file in the same directory as are the sorted BAM files. We can prepare the following sbatch SLURM file in nano text editor with a file name of samtools-index.slurm, which is saved in the same directory as the sorted BAM files.

```
#!/bin/bash

# SAMtools indexing of the sorted BAM files

#SBATCH --job-name=BAMindexing
#SBATCH --partition=express
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=5G
#SBATCH --time=0:30:00
#SBATCH --mail-type=FAIL
#SBATCH --mail-user=kejinhu@uab.edu
#SBATCH --error=BAMindex.err
#SBATCH --out=BAMindex.out

module load SAMtools

for i in *sortedByCoord.bam; do samtools index -b $i; done
```

Again, this code uses the for loop to conduct the repetitive tasks for different samples. After preparing the above sbatch SLURM file in the nano text editor, you can submit it using sbatch in the same directory the script file is located,

```
sbatch samtools-index.slurm
```

Please note that like samtools sort the samtools index command also take the --thread (-@) option. When we increase CPU from one (default) to 24 (samtools index -@ 24 or --thread=24), Cheaha takes 1 minutes only to complete the indexing of the 7 sorted BAM files (vs 7 minutes).

## 6.5 Briefly review the BAM files

You can see the alignments (SAM file contents) for any of your BAM files using the samtools view command,

srun --pty /bin/bash # This allocates you with compute resources using the default options. When you still have resources after the indexing process, you do not need to request again. When you close the pseudo terminal shell window for the allocated compute nodes or even the dashboard window (not the browser), it does not give up the allocated resources. In this case, you can go back to the clusters via the Clusters tab on the HPC dashboard and find out the job ID using squeue -u username, and then re-establish the pseudo terminal shell for the allocated nodes using, srun --jobid [ID#] --pty /bin/bash. If you want to give up the resources, use the exit command. When you close the browser (not the clusters window nor the dashboard window), you will also lose your compute nodes.

module load SAMtools

samtools view my.sortedByCoord.bam | head # This displays the first 10 alignments without the header information of the BAM file as a standard output in the SAM format. Please note that you can view the BAM and sorted BAM files using the samtools view command, but not the .BAI file because the .BAI file is the index file of the BAM file without the sequence data.

If you use the header option -h for the samtools view command, you can see the header information of each BAM file as the SAM output. Let us display the first 210 lines of your BAM file in the SAM format,

samtools view -h my.sortedByCoord.bam | head -n 210 # This displays on your screen the first 210 lines of the BAM file starting from its header information. As you can see, the alignment, sorting software and codes are also included in the header information (in the @PG lines and @CO lines). The header of a BAM file includes information about the entire file, such as version number (VN) of the SAM specification and sorting status (in the @HD line), reference sequence name and length (SN and LN in the @SQ lines), as well as alignment and sorting methods and codes. SN indicates names of chromosomes or contigs.

After indexing, you can also check the alignments in a specific region of a chromosome using the samtools view command,

samtools view my.sortedByCoord.bam 4:20100000-20180000 # This prints out the read alignments between 20100000 and 20180000 on chromosome 4 on your screen (standard output).

You can view the alignments line by line or page by page using the less command in combination with the samtools view command,

samtools view my.bam | less # After issuing this command you can use the down- and up-arrow keys, or page down and up keys to view the files. You can view the sorted or unsorted BAM files using the samtools view command. The file is long, and you can exit the file and return to the shell prompt line by hitting the q key.

You can check the simple statistics for a BAM file, samtools idxstats my.sortedByCoord.bam        # This code returns a statistic table (standard output) on your screen (the default destination of your stdout) for the target file with 4 columns in the following order: reference sequence names (chromosome or contig IDs), sequence length, number of the mapped reads/segments, and number of the unmapped reads/segments.

If you want to save the statistic output as a file and examine the results using other tools, you can use the standard output (stdout) redirection operator >,

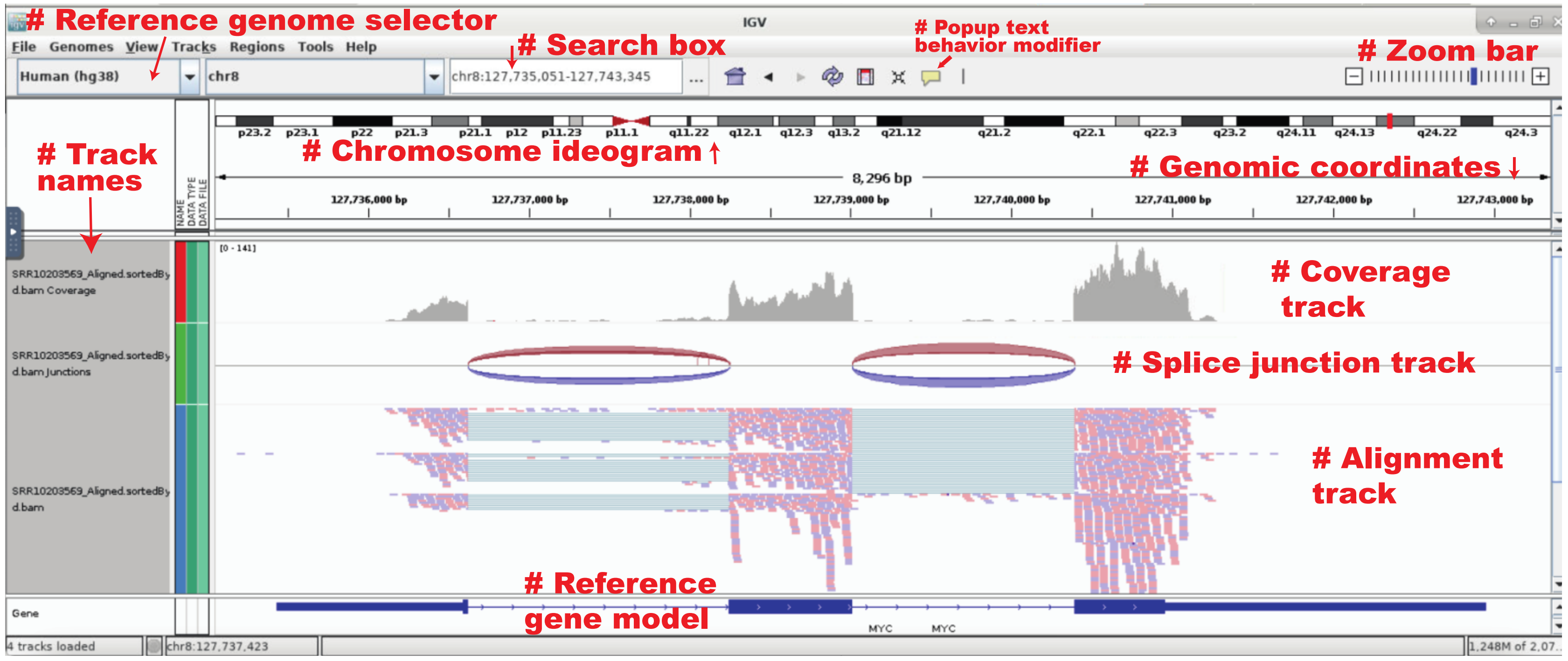
samtools idxstats my.sortedByCoord .bam > my.sortedBCoord.bam_stats        # This saves the stdout in the file of my.sortedByCoord.bam.stats in the same directory.

## 6.6 Visual inspection of your alignments with IGV

You can visually review the tracks of the alignments using the IGV (Integrative Genomics Viewer) tool. IGV is an open-source free software, which can be used on iMac, Windows and Linux with a web version [16-18]. Your HPC may have an IGV server, and you can conveniently examine your BAM files without transferring the BAM files to your local computers. If your HPC does not have an IGV server, you can view the alignment using your BAM files on your desktop IGV tool. Your BAM files need to be sorted and indexed in order to be viewed with IGV.

First, you request an interactive IGV app from your HPC dashboard (see the app list in Figure 34) with the default or customized resources. When your IGV resource is granted, launch it, and you will see the IGV app window. Click on the File tab to bring about the pull-down menu, and then click on the “Load from file” button. A new window interface pops up. Find the directory of your BAM files via the FileSystem. Go to the directory containing the BAM files, and select the sorted BAM file to load. Please note that your corresponding index file (with the .bai extension) should be in the same directory of your parent sorted BAM files. This will automatically create three tracks for each BAM file, coverage track, splice junction track, and alignment track (Figure 31).

You can examine the alignment for a specific chromosome, a region of a chromosome, or a specific gene by specifying the chromosome, chromosome region, or a gene symbol in the text search box (Figure 31).  For convenience, you can define the popup text behavior in data panels: “Show Details on Hover”, “Show Details on Click”, or “Never Show Details”. You can zoom in and out the alignments using the + and – buttons at the top-right corner (Figure 31). You can navigate along the coordinates by dragging the tracks (leftward or rightward). Desktop IGV works similarly.



# Reference genome selector
# Search box
# Popup text behavior modifier
# Zoom bar
# Track names
# Chromosome ideogram
# Genomic coordinates
# Coverage track
# Splice junction track
# Alignment track
# Reference gene model
IGV
File Genomes View Tracks Regions Tools Help
Human (hg38)
chr8
chr8:127,735,051-127,743,345
8,296 bp
SRR10203569_Aligned.sortedByd.bam Coverage
SRR10203569_Aligned.sortedByd.bam Junctions
SRR10203569_Aligned.sortedByd.bam
Gene
MYC
4 tracks loaded
chr8:127,737,423

**Figure 31**. Visual inspection of RNA-seq alignment results. Sample SRR10203569 is used, and the MYC gene is inspected here. Please note that there are two colors in the alignment track (blue and red) indicating use of the non-stranded protocol of RNA-seq. Some IGV components are illustrated in red texts.

## Summary of chapter 6

- The aligned reads should be sorted and indexed before counting.
- We use the SAMtools command samtools sort to sort and samtools index to index the BAM files.
- SAMtools also supports multiple threading. The sorting and indexing speed can be enhanced by multiple threading via the option of --thread.
- Indexing is fast but sorting is much slower. Sorting can also be sped up using bash array script. Array and multiple threading can be combined to reach the maximum speed of sorting.
- We can use the samtools view command to briefly review the aligned BAM files.
- The sorted and indexed BAM files can be reviewed visually using the IGV tool.

# Chapter 7

# Counting the reads to features using HTSeq

## 7.1 Find out the strandedness of RNA-seq data using IGV

To count the reads using the htseq-count command of the HTSeq module, we need to define the strandedness of your sequencing method. In fact, you can find out such a sequencing method without asking the sequencing facility. One simple way to reveal sequencing strandedness is to use IGV. In the IGV alignment track (the bottom track in Figure 31), right click your mouse to bring about the popup menu. In the popup menu, bring about the "color alignment by" sub-menu and choose "first-of-pair strand". This will highlight the aligned reads with two colors, each for one strand. Please note that you need to uncheck the "shade base by quality" and "show mismatched bases" to show the strand colors only. It is stranded sequencing if you see only one color for the aligned reads to a specific gene; it is not stranded sequencing if you see two colors in the aligned reads to a gene (the MYC gene as an example in Figure 31). You need to choose the correct version of your reference genome in IGV that agrees with the version of the reference genome used for the alignment process. Otherwise, the reference coordinates may not match the RNA-seq alignment. In this tutorial, we use hg38, not hg19 (selectable from the box in the upper-left corner in Figure 31).

## 7.2 Count the reads to features using the htseq-count command (serial counting)

Counting and alignment/mapping are the two processes that are time consuming. When the script is not optimized, it takes more than one hour to count one file/treatment even on Cheaha (for example, using the regular setting of htseq-count with great resources of 24 CPUs and 10 G per CPU, it took 20 hours and 27 minutes to count the 12 human RNA-seq samples with 30-40 million reads per sample. Each sample needs around 102 minutes to count.). Therefore, you may use the "medium" value for the "partition" option of SLURM, and request more than one day of time (--time option) to count more than 10 human RNA-seq samples when the code is not optimized. Sections 7.2 to 7.4 will introduce basic syntax of the htseq-count command and methods for efficient counting.

In the directory with the sorted and indexed BAM files, open the nano text editor with a file name of HTSeq_count.slurm,

```
nano HTSeq_count.slurm
```

In the nano text editor, prepare the following code to count the 7 samples in this tutorial using a bash for loop code,

```
#!/bin/bash
# HTSeq serial counting
```

```
#SBATCH --job-name=HTSeq_Serial_Count
#SBATCH --partition=medium
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=1G
#SBATCH --time=14:00:00
#SBATCH --mail-type=FAIL
#SBATCH --mail-user=hukejin@gmail.com
#SBATCH --error=count.err
#SBATCH --out=count.out

module load HTSeq

for i in *Aligned.sortedByCoord.bam; \
do htseq-count -f bam -s no -r pos $i \
/data/user/kejinhu/RNA_seq_tutorial/Homo_sapiens.GRCh38.105.gtf > $i.count; \
done
```

Then, in the directory the HTSeq_count.slurm file is located, submit the job,

```
sbatch HTSeq_count.slurm
```

# It took 7 hours and 32 minutes to count the 7 samples in this tutorial using the resources above (1 CPU, 1 G of memory). Increasing CPUs and memory did not improve the counting, for examples, using 12 CPU and 8 G per CPU it took around 8 hours and 5 minutes to count the 7 samples in this tutorial. This is because the default number of CPUs is one and there is no data level multiple threading for the htseq-count function. There is a parallel counting option with the command of htseq-count command, but it is at the sample level. When using the -n (--nprocesses) option to implement parallel counting, you will have a single output count table with counts of each sample as a column. This will need different protocol/procedure for statistics analyses from that described in Chapter 8 below.

This code uses the for loop to count all the sorted BAM files with the common ending of Aligned.sortedByCoord.bam. The syntax of the for loop is "for variable in character string; do application code; done". Here, "i' is the variable; "*Aligned.sortedByCoord.bam' is the character string. Please note that the path to the BAM files is omitted here because the HTSeq_count.slurm file and the BAM files are located in the same directory, and we will submit this job in this directory. If you submit the job in other directory, you should define the path to the BAM files. The resulting count files will be saved in the same directory of the source BAM files with an file extension of .count. The -f option defines the input file type, and the default is SAM. Here, we define -f as BAM file. The -s option specifies the strandedness of RNA sequencing method and the default value is *yes*. Here, the sample RNA-seq was sequenced using non-stranded library preparation method. The -r option defines the sorting method, and the *pos* value means the BAM files are sorted by position or coordinates. The htseq-count command requires the index GTF or GFF files and its full path should be specified unless you submit the job in the directory where the GTF file is located. Please note that BAM index (with the .bai extension) file is needed for the htseq-count command, and should be located in the same directory as the corresponding sorted BAM files. You index the bam file using the samtools index command as introduced earlier. The \ is the continuation operator to break long command line into two lines. > here is the re-direct operator, which divert the standard output into the file name specified after >. Alternatively, we can use the -c (count output file) option to specify the count output file names. In this case, you remove the "> $i.count" part and add the "-c $i.count" option for the htseq-count command.

For the details about usage of the htseq-count command, you can load the HTSeq module and use the help option for the htseq-count command, module load HTSeq, and then htseq-count -h. Please note that the package name is case sensitive when you load HTSeq. Please check your spelling if you cannot load the HTSeq module.

## 7.3 Simultaneous counting of multiple samples using srun

Counting is time consuming. The codes in the above section use the for loop to count the 7 samples one after one, i.e., serial counting. Since counting of each sample takes more than 1 hour, the total time for counting all the samples one after one is long. We can shorten the total time by counting all the 7 samples simultaneously. To this end, we combine the sbatch and srun commands. The srun command can be used within a sbatch script to define job steps. Open the nano text editor with a file name of parallel_count.slurm,

```
nano parallel_count.slurm
```

Then prepare the following code in the nano text editor,

```
#!/bin/bash
# HTSeq parallel counting

#SBATCH --job-name=ParallelCounting
#SBATCH --partition=express
#SBATCH --nodes=1
#SBATCH --ntasks=7
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=1G
#SBATCH --time=2:00:00
```

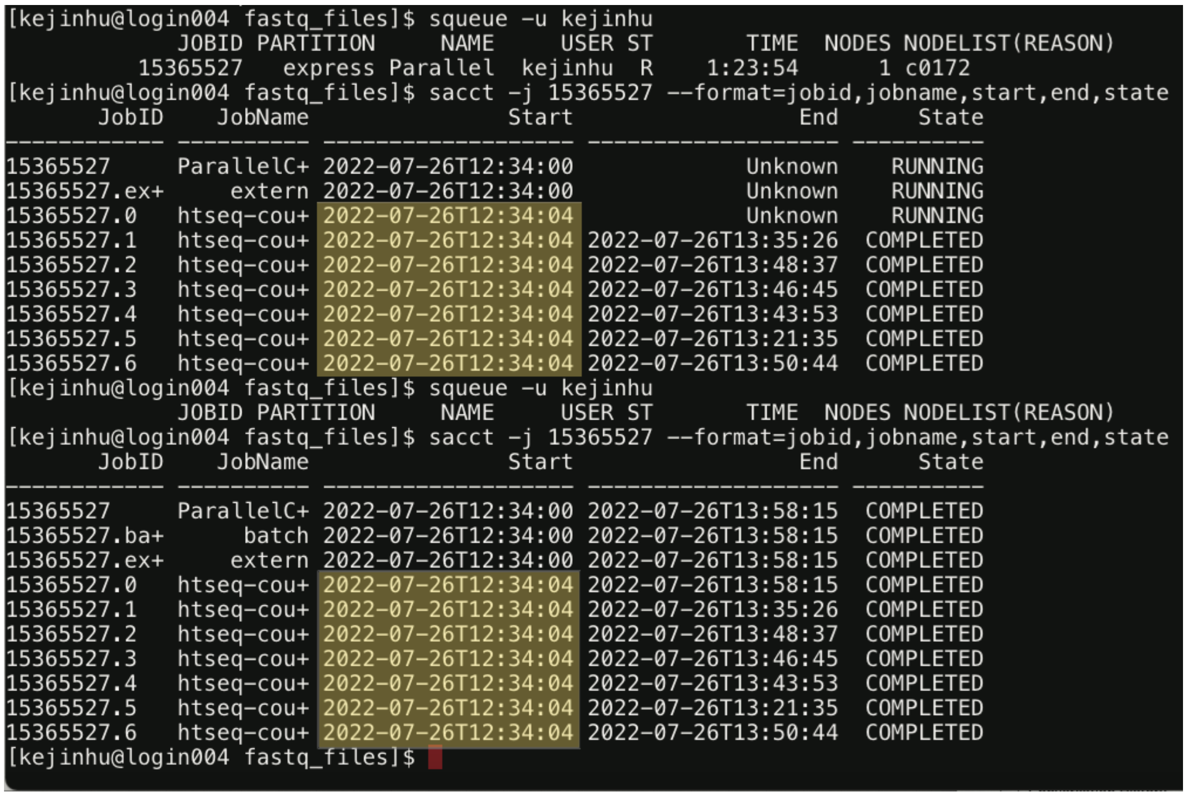
```
#SBATCH --mail-type=FAIL
#SBATCH --mail-user=kejinhu@uab.edu
#SBATCH --error=count.err
#SBATCH --out=count.out

module load HTSeq

srun -n 1 -c 1 --exclusive htseq-count -f bam -s no -r \
pos    -c    SRR10203569.count    SRR10203569_Aligned.sortedByCoord.bam    \
/data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/Homo_sapiens.\
GRCh38.105.gtf &

srun -n 1 -c 1 --exclusive htseq-count -f bam -s no -r pos \
-c SRR10203569.count SRR10203570_Aligned.sortedByCoord.bam \
/data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/Homo_sapiens.\
GRCh38.105.gtf &

srun -n 1 -c 1 --exclusive htseq-count -f bam -s no -r pos \
-c SRR10203569.count SRR10203571_Aligned.sortedByCoord.bam \
/data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/Homo_sapiens.\
GRCh38.105.gtf &

srun -n 1 -c 1 --exclusive htseq-count -f bam -s no -r pos \
-c SRR10203569.count SRR10203572_Aligned.sortedByCoord.bam \
/data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/Homo_sapiens.\
GRCh38.105.gtf &

srun -n 1 -c 1 --exclusive htseq-count -f bam -s no -r pos \
-c SRR10203569.count SRR10203573_Aligned.sortedByCoord.bam \
/data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/Homo_sapiens.\
GRCh38.105.gtf &

srun -n 1 -c 1 --exclusive htseq-count -f bam -s no -r pos \
-c SRR10203569.count SRR10203574_Aligned.sortedByCoord.bam \
/data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/Homo_sapiens.\
GRCh38.105.gtf &

srun -n 1 -c 1 --exclusive htseq-count -f bam -s no -r pos \
-c SRR10203569.count SRR10203575_Aligned.sortedByCoord.bam \
/data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/Homo_sapiens.\
GRCh38.105.gtf &
wait
```

In the above code, we use 7 srun commands to define 7 job steps, each for counting one sample. Please note that at the end of each job step, we use an ampersand sign & to send the job step to the background and the next job step can start immediately. In other words, the ampersand sign & removes the blocking feature of the srun command. At the end of the last job step, we use a wait command so that the slowest job step can be finished, i.e., no job steps will be canceled when the fast job steps have been completed. The above codes are similar to that when you request 7 individual pseudo shell terminals and run the htseq-count command on each pseudo terminal simultaneously. Please note that we use the continuation operator in each srun command because the command line is too long. The syntax for the htseq-count command is otherwise the same as that in the serial job. The --exclusive option means "don't share CPUs for job steps".

Please note that the total number of job steps (7 here) should be equal to the total number of tasks defined by the #BATCH --ntasks tag.

Then, submit the job,

```
sbatch parallel_count.slurm
```



```
[kejinhu@login004 fastq_files]$ squeue -u kejinhu
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          15365527   express Parallel  kejinhu  R    1:23:54      1 c0172
[kejinhu@login004 fastq_files]$ sacct -j 15365527 --format=jobid,jobname,start,end,state
       JobID    JobName               Start                 End      State
------------ ---------- ------------------- ------------------- ----------
15365527      ParallelC+ 2022-07-26T12:34:00             Unknown    RUNNING
15365527.ex+      extern 2022-07-26T12:34:00             Unknown    RUNNING
15365527.0    htseq-cou+ 2022-07-26T12:34:04             Unknown    RUNNING
15365527.1    htseq-cou+ 2022-07-26T12:34:04 2022-07-26T13:35:26  COMPLETED
15365527.2    htseq-cou+ 2022-07-26T12:34:04 2022-07-26T13:48:37  COMPLETED
15365527.3    htseq-cou+ 2022-07-26T12:34:04 2022-07-26T13:46:45  COMPLETED
15365527.4    htseq-cou+ 2022-07-26T12:34:04 2022-07-26T13:43:53  COMPLETED
15365527.5    htseq-cou+ 2022-07-26T12:34:04 2022-07-26T13:21:35  COMPLETED
15365527.6    htseq-cou+ 2022-07-26T12:34:04 2022-07-26T13:50:44  COMPLETED
[kejinhu@login004 fastq_files]$ squeue -u kejinhu
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
[kejinhu@login004 fastq_files]$ sacct -j 15365527 --format=jobid,jobname,start,end,state
       JobID    JobName               Start                 End      State
------------ ---------- ------------------- ------------------- ----------
15365527      ParallelC+ 2022-07-26T12:34:00 2022-07-26T13:58:15  COMPLETED
15365527.ba+       batch 2022-07-26T12:34:00 2022-07-26T13:58:15  COMPLETED
15365527.ex+      extern 2022-07-26T12:34:00 2022-07-26T13:58:15  COMPLETED
15365527.0    htseq-cou+ 2022-07-26T12:34:04 2022-07-26T13:58:15  COMPLETED
15365527.1    htseq-cou+ 2022-07-26T12:34:04 2022-07-26T13:35:26  COMPLETED
15365527.2    htseq-cou+ 2022-07-26T12:34:04 2022-07-26T13:48:37  COMPLETED
15365527.3    htseq-cou+ 2022-07-26T12:34:04 2022-07-26T13:46:45  COMPLETED
15365527.4    htseq-cou+ 2022-07-26T12:34:04 2022-07-26T13:43:53  COMPLETED
15365527.5    htseq-cou+ 2022-07-26T12:34:04 2022-07-26T13:21:35  COMPLETED
15365527.6    htseq-cou+ 2022-07-26T12:34:04 2022-07-26T13:50:44  COMPLETED
[kejinhu@login004 fastq_files]$
```

**Figure 32**. Parallel jobs start at the same time and run simultaneously. The start times for all steps are highlighted by shading. Please note that each job step may end at different times (see the End column).

You will see there is one SBATCH job ID when you issue, squeue -u [username] (Figure 32). Check this job using the following code,

```
sacct -j [jobID] --format=jobid,jobname,start,end,state
```

# The sacct command is used to display information about jobs and job steps (Figure 32). The different values for the --format option are separated by a comma without space between values. The available values for the --format option can be found using sacct --helpformat, which returns a list of the format values. Please note that --helpformat here is one word without space in between. The job ID option -j carries a value of the job number.

From the output of the above code, you will see each job step has its own ID, and that each job step starts at the same time (the Start column of the output table) (Figure 32). Therefore, counting of the 7 samples is conducted simultaneously, and the SBATCH job can be completed in much less time (around 84 minutes vs > 8 hours using the serial counting). However, each job step (each sample) ends at different time (the End column of Figure 32). The completion time of each sample is permanently recorded and can also be found by listing all the count files after counting, ls -hl *.count.

## 7.4 Run parallel counting of samples by job array

Alternatively, we can use the --array flag to run parallel counting of all the samples of a project. Prepare the following scripts in nano text editor with a file name of count_array.slurm,

```
#!/bin/bash
# HTSeq parallel counting using array
#SBATCH --job-name=HTSeqParallelCounting
#SBATCH --partition=express
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=7
#SBATCH --mem-per-cpu=1G
#SBATCH --time=2:00:00
#SBATCH --mail-type=FAIL
#SBATCH --mail-user=hukejin@gmail.com
#SBATCH --error=count.err
#SBATCH --out=count.out
#SBATCH --array=0-6

module load HTSeq
```

```
FILE=("SRR10203569Aligned.sortedByCoord.bam"
"SRR10203570Aligned.sortedByCoord.bam"
"SRR10203571Aligned.sortedByCoord.bam"
"SRR10203572Aligned.sortedByCoord.bam"
"SRR10203573Aligned.sortedByCoord.bam"
"SRR10203574Aligned.sortedByCoord.bam"
"SRR10203575Aligned.sortedByCoord.bam")

htseq-count -f bam -s no -c ${FILE[$SLURM_ARRAY_TASK_ID]}.count \
-r pos ${FILE[$SLURM_ARRAY_TASK_ID]} \
/data/user/kejinhu/RNA_seq_tutorial/humanGenomeIndex/\
Homo_sapiens.GRCh38.105.gtf
```

This code uses the --array flag to define the number of parallel jobs, and the FILE bash variable to define the 7 samples. The SLURM_ARRAY_TASK_ID is used to define each sample to be counted and saved.

Then, submit the file,

```
sbatch count_array.slurm
```

This array parallel counting and the above srun parallel counting used similar amount of time to count the 7 samples.

## 7.5 Briefly review the counted results in the terminal

The above htseq-count command returns one .count file for each of the BAM file. Use the head command to see the first 10 features of the count table (Figure 33),

```
head SRR10203569_Aligned.sortedByCoord.bam.count          # Please note that
the count file names are shortened (re-named) in Figure 33 (see Chapter 8 below).
```

As you can see that the htseq-count command generates a 2-column table for each BAM file. The first column is the ENSEMBL ID, and the second column is the read counts uniquely mapped to a specific ENSEMBL ID (Figure 33).

Use the tail command to see the last 10 lines of a count file (Figure 33),

```
tail SRR10203569_Aligned.sortedByCoord.bam.count   # Please note that the count
file names are shortened (re-named) in Figure 33 (see Chapter 8 below).
```

You can see that at the end of the table there are summary for other categories of read counts, which include _no_feature, _ambiguous, _too_low_aQual, _not_aligned, and _alignment_not_unique (lower part of Figure 33). Please make sure that the read counts for those categories are very low, and the majority of read counts are uniquely mapped features.

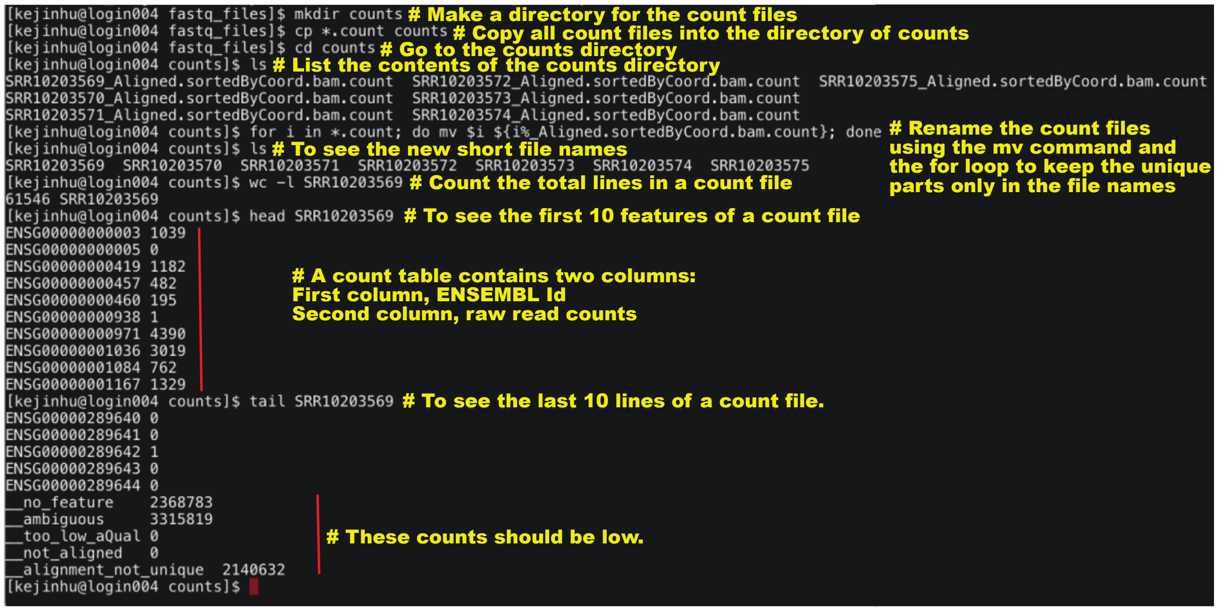
You can check the entire count table using the less command,

> less SRR10203569_Aligned.sortedByCoord.bam.count # After the less command, you can use down- and up-arrow keys, or page-down and -up keys to navigate down and up the table. Instead of the less command, you can also use the cat command to see their contents of the count files.

You can find out how many row/transcripts in a specific or all the count files using the wc (Word Count) command (Figure 33),

> wc -l SRR10203569_Aligned.sortedByCoord.bam.count # This counts the number of lines for this count file. The -l option means counting the lines in the table.

> wc -l *.count # This counts number of lines for each of the count files in the same directory. Usually, all count files have the same number of lines.



```
[kejinhu@login004 fastq_files]$ mkdir counts # Make a directory for the count files
[kejinhu@login004 fastq_files]$ cp *.count counts # Copy all count files into the directory of counts
[kejinhu@login004 fastq_files]$ cd counts # Go to the counts directory
[kejinhu@login004 counts]$ ls # List the contents of the counts directory
SRR10203569_Aligned.sortedByCoord.bam.count  SRR10203572_Aligned.sortedByCoord.bam.count  SRR10203575_Aligned.sortedByCoord.bam.count
SRR10203570_Aligned.sortedByCoord.bam.count  SRR10203573_Aligned.sortedByCoord.bam.count
SRR10203571_Aligned.sortedByCoord.bam.count  SRR10203574_Aligned.sortedByCoord.bam.count
[kejinhu@login004 counts]$ for i in *.count; do mv $i ${i%_Aligned.sortedByCoord.bam.count}; done # Rename the count files
                                                                                                  using the mv command and
                                                                                                  the for loop to keep the unique
                                                                                                  parts only in the file names
[kejinhu@login004 counts]$ ls # To see the new short file names
SRR10203569  SRR10203570  SRR10203571  SRR10203572  SRR10203573  SRR10203574  SRR10203575
[kejinhu@login004 counts]$ wc -l SRR10203569 # Count the total lines in a count file
61546 SRR10203569
[kejinhu@login004 counts]$ head SRR10203569 # To see the first 10 features of a count file
ENSG00000000003 1039
ENSG00000000005 0
ENSG00000000419 1182
ENSG00000000457 482          # A count table contains two columns:
ENSG00000000460 195          First column, ENSEMBL Id
ENSG00000000938 1            Second column, raw read counts
ENSG00000000971 4390
ENSG00000001036 3019
ENSG00000001084 762
ENSG00000001167 1329
[kejinhu@login004 counts]$ tail SRR10203569 # To see the last 10 lines of a count file.
ENSG00000289640 0
ENSG00000289641 0
ENSG00000289642 1
ENSG00000289643 0
ENSG00000289644 0
__no_feature    2368783
__ambiguous     3315819
__too_low_aQual 0                  # These counts should be low.
__not_aligned   0
__alignment_not_unique  2140632
[kejinhu@login004 counts]$
```

**Figure 33**. Management and review of the count tables.

After you transfer these count files to your local device, you can open the table using LibreOffice Calc (Linux) or Excel (MAC or Windows PC). If around 50% of your reads are in the group of no_feature when you use the default value for the strandedness option -s yes, your samples are likely prepared using the non-stranded library kit. The total aligned read counts can be found in the _Log.final.out files. In this case, you need to count again using the -s no option for the htseq-count command.

## Summary of Chapter 7

- This tutorial uses the htseq-count command to count reads to features.
- Counting of reads to features is very slow.
- We can use srun in a sbatch file to achieve parallel counting of multiple samples and significantly shorten the machine time for counting.
- Bash job array can also be used to speed up counting by processing many samples simultaneously.

# Chapter 8

# Analyses for differential expression using the DESeq2 R package

## 8.1 Preparation of count files on Linux HPC

Each command in this section takes no time and uses very little resources, and therefore we may conduct such file-management work on the login node. But, it is better to start a pseudo terminal using srun --pty bash even for this light job. Usually, I put all count files in one directory called counts, using the cp (copy) command,

```
cd /data/user/kejinhu/RNA_seq_tutorial/fastq_files   # This takes you to the
directory where the read count files are currently located.

mkdir counts          # This makes a subdirectory of counts (Figure 33).

cp *.count counts     # This copies all the count files into the directory of counts
(Figure 33).
```

Then, go into the *counts* directory,

```
cd counts
```

Within the *counts* directory, we remove the common part of the count file names, i.e., the "_Aligned.sortedByCoord.out.bam.count" part from all count files, but keep the unique sample identity parts, using the following for loop in combination with the mv (move) command (Figure 33),

```
for i in *_Aligned.sortedByCoord.out.bam.count; \
do mv $i ${i%_Aligned.sortedByCoord.out.bam.count}; done   # The % sign here
means that the substring after it will be deleted from the variable (Figure 33).
```

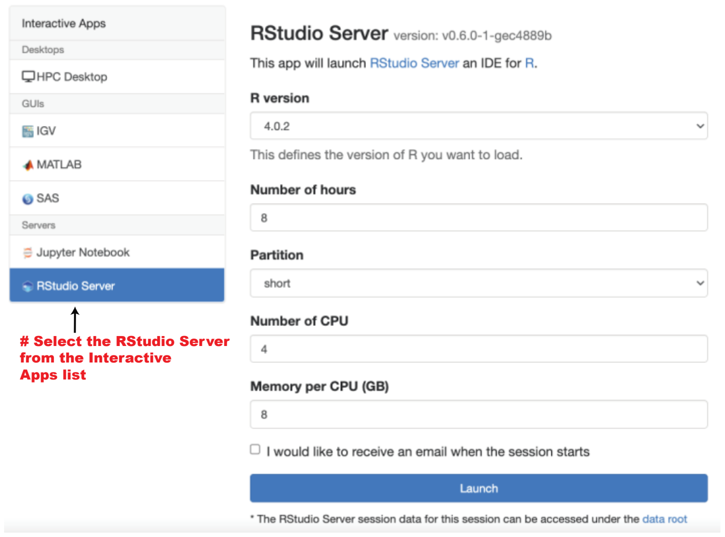
Now, you can use the ls command to see the new file names in the *counts* directory (Figure 33). As you can see, the above codes shorten the file names and generate 7 new count file names of SRR10203569, SRR10203570, SRR10203571, SRR10203572, SRR10203573, SRR10203574, SRR10203575 (Figure 33).

## 8.2 Request resources of RStudio server

Bring about the pulldown menu at the Cheaha Interactive Apps tab (Figure 2) on the HPC web dashboard, then click the RStudio Server tab from the menu, and this gives you the request form (Figure 34). Click the Launch button if you will not change any parameters (using the default settings). You will see a queue window. After the requested resource is granted, click the "® Connect to RStudio Server" button, and this will take you to the RStudio platform.

For the procedure below you can do all the analyses on RStudio of your desktop if you do not have a HPC RStudio server. However, you need to install all the required R packages on your local computer. Otherwise, the procedure is very similar. Installation of R and RStudio is easy and straightforward, and you can seek for help from your institution IT personnel. For the basics of R, readers can consult some introductory books [8, 9], or my tutorials [6, 7]. In those two tutorials I introduce some basic concepts and commands related to preparation of heat maps and boxplots. In this tutorial below, I will further introduce some basic as well as the RNA-seq related R commands.

## 8.3 General preparation in RStudio for DESeq analyses



# Select the RStudio Server from the Interactive Apps list

RStudio Server version: v0.6.0-1-gec4889b

This app will launch RStudio Server an IDE for R.

R version: 4.0.2

This defines the version of R you want to load.

Number of hours: 8

Partition: short

Number of CPU: 4

Memory per CPU (GB): 8

☐ I would like to receive an email when the session starts

* The RStudio Server session data for this session can be accessed under the data root

**Figure 34**. Request resources for RStudio server.

On the RStudio console pane, you will see a prompt sign > (The prompt sign in Linux is $, compare Figures 32 and 33 with 35). Type getwd(), and then hit the Enter key. You will see your current working directory is your data home directory of your HPC account. In my case, it is /home/kejinhu/ (Figure 35). As you can see here, unlike Linux an R command (in R terminology, it is called function) uses parentheses to introduce its arguments (Linux terminology is options). You should use the parentheses even though no arguments are specified (i.e., default) as you can see here for getwd().

You may see some R objects listed on the Environment pane (upper right quadrant). This is because RStudio will automatically load the workspace saved as a hidden file, .RData in the home directory. Issue ls(), and you may see R objects you generated previously. Please note the different formats of the list function of R ls(), and the list command of Linux ls. If there is no any object, you will see “character(0)” on your screen. If there is any object, you can find the .RData file in your home directory using the list.files() function, list.files(all.files = TRUE)          # If you do not use the all.files = TRUE argument, i.e., list.files(), the hidden file .RData will not show.

You can remove all the objects (not the files) using the following code,

rm(list=ls())          # This removes all the previous objects loaded onto the memory. You can call ls() again, and you will see a return of "character(0)" in your console, and the R objects will disappear from the "Environment" pane. You can also click "clear workplace" under the *Session* sub-menu to remove all objects on the memory.

The command rm(list=ls() just removes all the objects on the memory, but does not remove any files including the .RData file containing the objects. You can re-load this file again manually using the load function load(),

load(".RData")   # This re-loads the workplace. The file may have a specific name with the extension of .RData if you specify it at the time of saving.

ls()                    # This prints on the screen the loaded R objects from the memory.

rm(list=ls())           # Remove all the R objects on the memory again.

Now, set up your R working directory to the HPC directory that contains your RNA-seq count data using the following code with the set working directory function setwd(),

setwd("absolute_path_to_count_directory")          # In my case, the code is setwd("/data/user/kejinhu/RNA_seq_tutorial/fastq_files/counts") (Figure 35). Please note that you need to quote the path in the setwd() function. This will change your RStudio working directory to the *counts* directory. You can confirm the new working directory using the getwd() function.

getwd()          # This prints out the path to your *counts* directory on your HPC. In my case, it is /data/user/kejinhu/RNA_seq_tutorial/fastq_files/counts.

To see all the files in the working directory, issue,

list.files()      # You will see a list of files on your screen. In this tutorial, you will see the count file names, "SRR10203569" "SRR10203570" "SRR10203571" "SRR10203572" "SRR10203573" "SRR10203574" "SRR10203575" (Figure 35). Make sure all your count files are there.  Please note that there is an "s" in the list.files() function. You would see a warning if you missed it.

## 8.4 Generate the DESeq2 R objects for DE analyses

DESeq2 needs several R objects for the analyses depending on the pipeline you will use. For the htseq-count and DESeq2 pipeline, we need R objects of *directory*, *sampleFiles*, *sampleCondition*, and *sampleTable*. The following procedures use the generic R functions to generate the DESeq2 objects, but we do not need the DESeq2 package at this point.

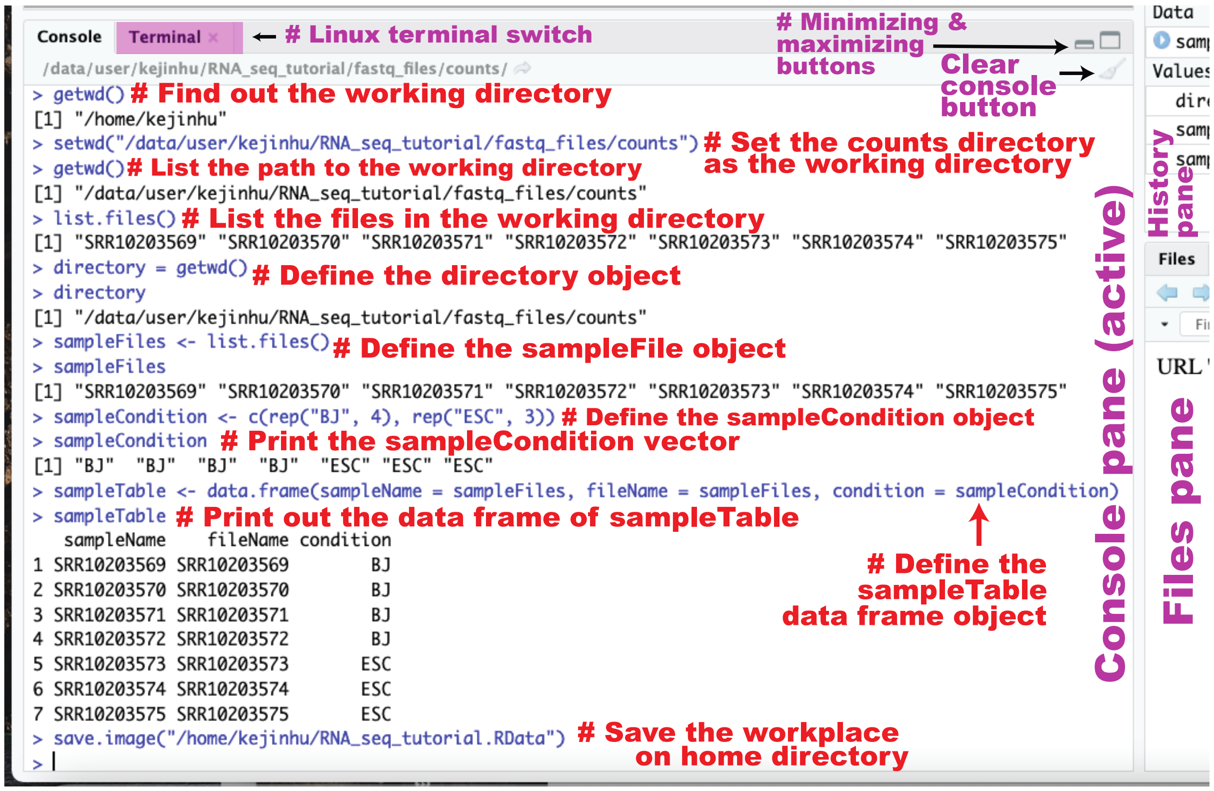

```
/data/user/kejinhu/RNA_seq_tutorial/fastq_files/counts/
> getwd()
[1] "/home/kejinhu"
> setwd("/data/user/kejinhu/RNA_seq_tutorial/fastq_files/counts")
> getwd()
[1] "/data/user/kejinhu/RNA_seq_tutorial/fastq_files/counts"
> list.files()
[1] "SRR10203569" "SRR10203570" "SRR10203571" "SRR10203572" "SRR10203573" "SRR10203574" "SRR10203575"
> directory = getwd()
> directory
[1] "/data/user/kejinhu/RNA_seq_tutorial/fastq_files/counts"
> sampleFiles <- list.files()
> sampleFiles
[1] "SRR10203569" "SRR10203570" "SRR10203571" "SRR10203572" "SRR10203573" "SRR10203574" "SRR10203575"
> sampleCondition <- c(rep("BJ", 4), rep("ESC", 3))
> sampleCondition
[1] "BJ"  "BJ"  "BJ"  "BJ"  "ESC" "ESC" "ESC"
> sampleTable <- data.frame(sampleName = sampleFiles, fileName = sampleFiles, condition = sampleCondition)
> sampleTable
   sampleName    fileName condition
1 SRR10203569 SRR10203569        BJ
2 SRR10203570 SRR10203570        BJ
3 SRR10203571 SRR10203571        BJ
4 SRR10203572 SRR10203572        BJ
5 SRR10203573 SRR10203573       ESC
6 SRR10203574 SRR10203574       ESC
7 SRR10203575 SRR10203575       ESC
> save.image("/home/kejinhu/RNA_seq_tutorial.RData")
> |
```
Console
Terminal
# Linux terminal switch
# Minimizing & maximizing buttons
Clear console button
# Find out the working directory
# Set the counts directory as the working directory
# List the path to the working directory
# List the files in the working directory
# Define the directory object
# Define the sampleFile object
# Define the sampleCondition object
# Print the sampleCondition vector
# Print out the data frame of sampleTable
# Define the sampleTable data frame object
# Save the workplace on home directory
Console pane (active)
History pane
Files pane
Data
Values
Files
URL

**Figure 35**. Prepare DESeq2 R objects in RStudio. The screenshot here mainly captured the Console pane and the other three panes (Plots/Files/Packages/Help pane, Source editor pane, and Workspace Browser/History pane) were largely avoided. Red texts are illustraions added by the author. The magenta texts illustrate the RStudio pane components. The Console pane can be minimized or maximized using the pane icons on the top right; the Console pane can be changed into a Linux terminal using the tab on the top left of the pane (shaded in red). On the Linux terminal pane, you can manipulate your files on the HPC using the bash commands.

directory = getwd()   # This generates the directory object for later use. Since we have made the *counts* directory as the working directory, we can use the getwd() function to assign elements to the directory object. We can also directly generate it using this syntax, directory = "absolute path to the directory of your count files". In my case, it is, directory = "/data/user/kejinhu/RNA_seq_tutorial/fastq_files/counts". Please note that you can use either = or <- assign operators. You can issue directory after the object is generated, and you can see the contents of the directory object, in the current case, "/data/user/kejinhu/RNA_seq_tutorial/fastq_files/counts" (Figure 35)

directory # This outputs the content of the object directory you just generated.

sampleFiles <- list.files() # The sampleFiles object can be generated in several ways. You can use: sampleFiles <- list.files(directory), or use the combine/concatenate c() R function, sampleFiles <- c("file1", "file2", "file3", ……, "fileN"). Please note that if there are other files in the same directory, you cannot use list.files() or list.files(directory) because these will include other non-target files as well. However, you can generate an R vector containing the names of all the files in the working directory using the list.files() function first, and then remove the unwanted elements using the index operator [ ], sampleFiles <- sampleFiles[-c(n1, n2)]. Here, -n1 and -n2 are the indices (position) of the unwanted elements of the sampleFiles vector, and the minus sign – here means to remove these files.

Let us simply generate the sampleFiles object using the R generic combine c() function,

sampleFiles # This prints the content of the object sampleFiles.

rm(sampleFiles) # This removes the sampleFiles object.

sampleFiles # You will see a warning of "Error: object 'sampleFiles' not found" because you just removed it.

sampleFiles <- c("SRR10203569", "SRR10203570", "SRR10203571", "SRR10203572", "SRR10203573", "SRR10203574", "SRR10203575") # This generates the sampleFiles object using the combine c() function.

sampleFiles # You will see you have just generated the same sampleFils object as does sampleFiles <- list.files().

sampleConditon <- c(rep("BJ", 4), rep("ESC", 3)) # Here, we use the c() function and rep() function to generate the sampleCondition object (a vector object). For a two-condition experiment, the syntax is sampleCondition <- c(rep("treatment", #_of_repeats), rep("control", #_of_repeats)). Use the help() function to find out information about the R functions of c() and rep(), i.e., help(c) and help(rep).

sampleTable <- data.frame(sampleName = sampleFiles, fileName = sampleFiles, condition = sampleCondition) # Here, we use the data.frame() function to generate the sampleTable object (a data frame object). For more information about the data.frame() function, you can call help(data.frame) on the RStudio console pane and see the help information on the Help pane. The resulting column names may not be sampleName, and it can be sampleNames; and fileName could be fileNames.

After generating the sampleTable data frame object, call the object sampleTable to print out the data frame on the screen and check if the sample names, sample files, and conditions match each other.

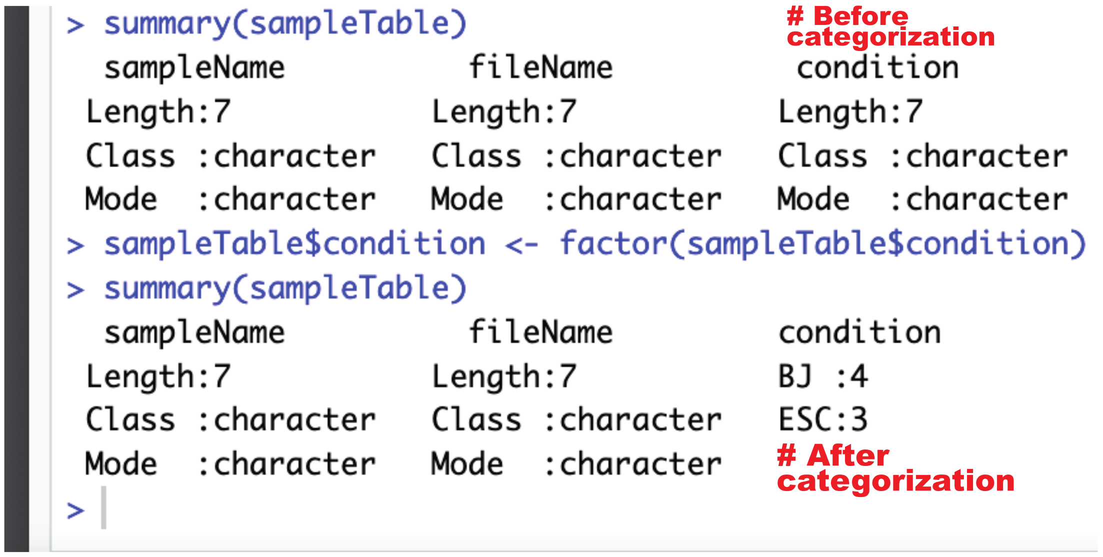
```
sampleTable
```

To see the basic features about the sampleTable object, use the summary() function (Figure 36),

```
summary(sampleTable)
```

Next, we categorize the condition vector of the sampleTable data frame as two-level factors using the factor() R function (Figure 36),

sampleTable$condition <- factor(sampleTable$condition)     # Here, we use the $ operator to select the condition column of the data frame of sampleTable. For use of the factor() function, you can issue help(factor) in your RStudio console pane and the help page for the factor() function will appear in the Help pane.



```
> summary(sampleTable)                                      # Before categorization
  sampleName          fileName           condition
 Length:7           Length:7           Length:7
 Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character
> sampleTable$condition <- factor(sampleTable$condition)
> summary(sampleTable)
  sampleName          fileName          condition
 Length:7           Length:7           BJ :4
 Class :character   Class :character   ESC:3
 Mode  :character   Mode  :character   # After categorization
>
```

**Figure 36**. Screenshot for categorization of the condition variables in the sampleTable data frame. Here, condition has two levels.

Use the summary() function again, and you will see that the condition column is categorized as a vector of factors, summary(sampleTable)

Your RStudio consoles screen may be cluttered with the codes and outputs. You can clear the console screen by clicking the "Clear Console" icon (upper right in Figure 35) or just use the key combination of ^+L (press Control and L together). This just clears the displays on the console screen and has no impact on the objects on the memory nor the function history.

## 8.5 Load DESeq2 package and get online help with DESeq2

When the objects directory, sampleFiles, sampleCondition, and sampleTable are defined, you can work under any other directory since the objects are on the memory and the paths to the files are defined already, but for convenience we will stay in the same directory. Now, we need the R package DESeq2 and its related packages. Please note that instructions about installation of R packages are not included in this tutorial. Load the DESeq2 library/package using the R function of library(),

library(DESeq2)            # When you load the DESeq2 library, R automatically loads the required packages for DESeq2 including S4Vectors, stats4, BiocGenerics, parallel, Biobase, and MatrixGenerics. These packages contain many objects with the same names as in the R base packages. Therefore, you may see warning for masking those objects. Do not worry about those warnings.

You can find the original article about DESeq2 using the citation() function,

citation("DESeq2")                    # Please note that you should quote DESeq2 in this code.

You can find out the version of your DESeq2 using the packageVersion() R function,

packageVersion("DESeq2")

Or

package.version("DESeq2")

For help with DESeq2, issue the following code,

help(DESeq)            # Please note that there are no quotation marks around DESeq becuae DESeq() is an R function as defined in the DESeq2 package, and there is no "2" because the function name is DESeq() not DESeq2(). This is similar to display help documents of other R functions such as c() and factor(), you issue help(c) or help("c") and help(factor) or help("factor") with and without quotation marks around the function name in query. This will give you documentation about the function DESeq() in the "Help" pane of RStudio. Unlike the help("DESeq2-package") introduced below, this call gives help documentation for the function of DESeq() only, but not other functions in the DESeq2 package.

If you want to get help about the DESeq2 package, using,

help("DESeq2-package")          # You will see a waring of "No documentation for 'DESeq2' in specified packages and libraries" when you use help(DESeq2) or help("DESeq2"). You will also have issues when you do not quote DESeq2-package here. You need to use help("DESeq2-package"). When you run this help, you will see brief information about the package DESeq2. There is a list of the main DESeq2 functions including DESeqDataSet(), DESeq(), lfsShrink(), and vst(). You can further run the ? or help() functions to find out information about these DESeq2 functions, e.g., help(DESeqDataSet).

However, you can find function help for the DESeq2-associated functions, for examples,

?DESeq2::counts          # Interestingly, you cannot use help(DESeq2::counts), or help("DESeq2::counts"). Please note that you need to use double colon sign between DESeq2 and counts.

?DESeq2::results          # Interestingly, you cannot use help(DESeq2::results), or help("DESeq2::results").

## 8.6 Conduct analyses with DESeq2

### 8.6.1 Prepare the raw DESeq data set (dds)

First, we need to generate the DESeq data set (dds) for analyses. The codes for dds generation vary depending on how you prepare the RNA-seq count tables in the previous step. We have used the htseq-count command to generate the count tables, and we therefore use the function DESeqDataSetFromHTSeqCount() to generate the DESeq data set (dds) (Figure 37),

```
ddsHTSeq <- DESeqDataSetFromHTSeqCount(sampleTable = sampleTable,
directory = directory, design = ~ condition)
```

You can retrieve the online help pages for the DESeqDataSetFromHTSeqCount() R function defined by DESeq2,

```
help(DESseqDataSetFromHTSeqCount)
```

Check the DESeqDataSet object ddsHTSeq (Figure 37), ddsHTSeq        # You will see a summary of the results including dimension, column names, and some row names.

At this point, you have a DESeqDataSet raw data. The count matrix of this dataset contains a lot of rows, and you can find the numbers of rows by calling the R function number of row nrow() (Figure 37),

nrow(ddsHTSeq)        # This prints out the total number of rows of the count matrix. You can find out the number of columns using the number of column R function ncol(), i.e., ncol(ddsHTSeq). You can find out the numbers of columns and rows at one time by calling the dimension function dim(), i.e., dim(ddsHTSeq). Use the colnames() function to print out the list of the column names, colnames(ddsHTSeq).

There are a lot of genes that are not expressed in any of the samples in your experiments. We can find out how many of these genes using the code below,

nrow(ddsHTSeq[rowSums(counts(ddsHTSeq)) < 1 ,])    # This outputs the number of genes/transcripts whose total read counts in all samples are 0. This code uses a combination of R functions rowSums(), counts(), nrow(), and the indexing operator [ ]. The comma in this code means all columns in the tables will be kept. Use of R function combination here is similar to the pipe operator | in Linux in that you use the output of one command as input of another command.

You may want to reduce the matrix table given that a lot of rows have a total read counts of zero or only one read for all the samples in the same row (a gene or a transcript). We are not interested in those genes without expression in both conditions. The following code removes all rows in which the total read counts for all samples are equal or less than 1 (Figure 37),

ddsHTSeq_noZero <- ddsHTseq[rowSums(counts(ddsHTSeq)) > 1 ,]        # This generates a smaller ddsHTSeq data set, containing rows with the corresponding total read counts for all samples greater than 1. Check the row and column numbers again by calling, dim(ddsHTSeq_noZero). You will see that the number of rows has been reduced significantly, but all the 7 columns are retained in the new ddsHTSeq data set (Figure 37).

### 8.6.2 Set up the reference level and run the DESeq() function

When conducting the differential expression analyses, you need one condition to be the reference, and this usually is the control condition. DESeq() will automatically choose the reference level alphabetically. You can define the reference level using the relevel() function and the ref argument.

ddsHTSeq_noZero$condition <- relevel(ddsHTSeq_noZero$condition, ref = "BJ") # Here, we use the human fibroblasts BJ cells as the reference. One usually uses the control as reference. The syntax is: ddsHTSeq$condition <- relevel(ddsHTSeq$condition, ref = "control").

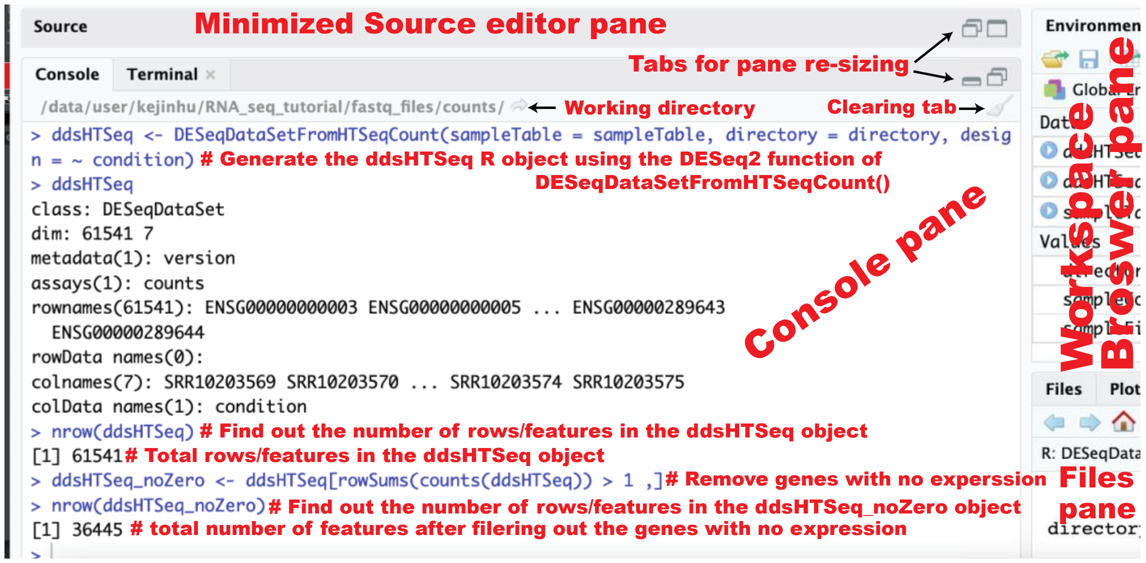
Then you can call the DESeq() command to generate the dds object,

dds <- DESeq(ddsHTSeq_noZero) # When you run this code you will see the message coming out one by one on your console screen,

```
estimating size factors
estimating dispersions
gene-wise dispersion estimates
mean-dispersion relationship
final dispersion estimates
fitting model and testing
```

When done, you can call,

resultsNames(dds) # This returns the names of the estimated effects (coefficients) of the model.



```
/data/user/kejinhu/RNA_seq_tutorial/fastq_files/counts/
> ddsHTSeq <- DESeqDataSetFromHTSeqCount(sampleTable = sampleTable, directory = directory, desig
n = ~ condition) # Generate the ddsHTSeq R object using the DESeq2 function of
                                                  DESeqDataSetFromHTSeqCount()
> ddsHTSeq
class: DESeqDataSet
dim: 61541 7
metadata(1): version
assays(1): counts
rownames(61541): ENSG00000000003 ENSG00000000005 ... ENSG00000289643
  ENSG00000289644
rowData names(0):
colnames(7): SRR10203569 SRR10203570 ... SRR10203574 SRR10203575
colData names(1): condition
> nrow(ddsHTSeq) # Find out the number of rows/features in the ddsHTSeq object
[1] 61541# Total rows/features in the ddsHTSeq object
> ddsHTSeq_noZero <- ddsHTSeq[rowSums(counts(ddsHTSeq)) > 1 ,]# Remove genes with no experssion
> nrow(ddsHTSeq_noZero)# Find out the number of rows/features in the ddsHTSeq_noZero object
[1] 36445 # total number of features after filering out the genes with no expression
>
```

**Figure 37**. Screenshot of RStudio pane for defining the ddsHTSeq object. It captured mainly the console pane with partials of the other three RStudio panes. Red texts are illustration by the author. R scripts/codes are in blue.

### 8.6.3 Extract results using the results() function

res <- results(dds)      # This generates a result table (stored in the result object res) with 6 columns: baseMean, log2FoldChange, lfcSE, stat, pvalue and padj, but does not include the read counts of each sample. You will use the counts() function to generate the count table (see below).

If your conditions have more than 2 levels, you may need to use the contrast argument to conduct pair-wise comparisons,

res_treat2_vs_control <- results(dds, contrast = c("condition", "treat2", "control"))      # This will extract a result table for the comparison between treatment 2 and the control, and store the results in the R object of res_treat2_vs_control. You can give any name for the result R object, but an informative name will be helpful.

### 8.6.4 Briefly review the results

You can use the summary() function to see the summary of the results,

summary(res)      # Or, you call summary(res_treat2_vs_control).

The above code returns a summary of the results. The outputs may look like this,

```
out of 36445 with nonzero total read count
adjusted p-value < 0.1
LFC > 0 (up): 8323, 23%
LFC < 0 (down): 6941, 19%
outliers [1]: 32, 0.088%
low counts [2]: 7773, 21%
(mean count < 1)
[1] see 'cooksCutoff' argument of ?results
[2] see 'independentFiltering' argument of ?results
```

You can print the content of the res object on your screen simply calling the object name,

```
res
```

Simply issue the res object as above will print out the abridged table with the head information similar to,

```
log2 fold change (MLE): condition ESC vs BJ
```

```
Wald test p-value: condition ESC vs BJ
DataFrame with 36445 rows and 6 columns
                    baseMean   log2FoldChange    lfcSE      stat        pvalue          padj
                   <numeric>   <numeric>     <numeric> <numeric>   <numeric>    <numeric>
ENSG00000000003  1598.7341      1.329407      0.239609    5.54824   2.88567e-08   1.68287e-07
ENSG00000000005    31.5038      8.657379      1.149997    7.52818   5.14537e-14   5.25172e-13
ENSG00000000419  1435.7501      0.879933      0.290701    3.02694   2.47045e-03   6.67570e-03
ENSG00000000457   298.9891     -0.929369      0.295471   -3.14538   1.65874e-03   4.65110e-03
ENSG00000000460   430.9958      2.208751      0.354075    6.23809   4.42942e-10   3.16513e-09
...                 ...         ...       ...     ...        ...        ...
ENSG00000289640   0.308377      1.95779      3.76297     0.520279    6.02869e-01        NA
ENSG00000289641   0.236623      1.69298      3.78028     0.447845    6.54265e-01        NA
ENSG00000289642   1.037582     -3.35650      2.37325    -1.414303    1.57273e-01    2.44347e-01
ENSG00000289643 128.914189     10.69002      1.14804     9.311529    1.26005e-20    2.10671e-19
ENSG00000289644   0.387676      2.26137      3.73836     0.604909    5.45239e-01        NA
```

When we use summary(res), statistic numbers for DE genes at the p<0.1 level are provided as above. We can find out how many genes/transcripts are differentially expressed at the padj<0.05 level (or at other levels) using a code (or similar) below,

sum(res$padj < 0.05, na.rm = TRUE)                # Please note that there are a lot of NA in the columns of pvalue and padj (see the output above). You will see a message of "NA" if you do not remove NA in your call as in this code: sum(res$padj < 0.05). You can find out how many NA in the padj columns: table(is.na(res$padj)), which gives a table listing number of the TRUE and FALSE events. Or, you just use sum(is.na(res$padj)) to print out the total number of NA. You can also find out how many rows in the padj columns of the res data frame are non-NA using this code, sum(!is.na(res$padj)).

Or, you can make the following call,

table(res$padj < 0.05)              # This generates a small table tabulating the number of TRUE and FALSE for the two categories.

You can see the first 6 rows/genes of your results table using the head() R function,

head(res)           # Or, you can see the specified number of rows via the n argument, for example, head(n=10, res). This will give you a sense about differences in using the head command of Linux and the R head() function.

### 8.6.5 Generate a table containing both statistics and counts

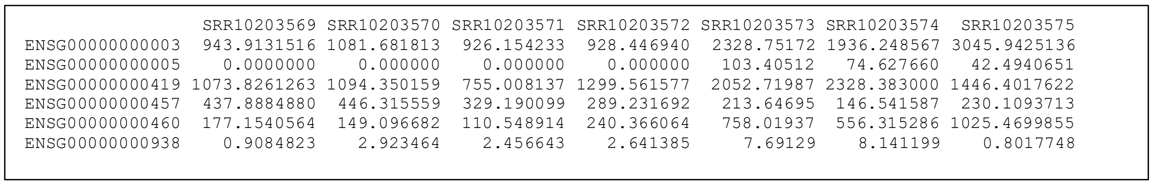
The results() function generates a table of 6 columns containing the statistical data only without the read count data. The read count table can be extracted using the function of counts(), count_table <- counts(dds, normalized = TRUE)

To see the first 6 rows/transcripts of the count table, issue,

head(count_table)        # An example of the output is shown in Figure 38.

The two tables can be combined using the cbind() R function,

res_count <- cbind(res, count_table)    # Or, you can generate the res_count table at one step combining the functions, res_count <- cbind(results(dds), counts(dds, normalized = TRUE)). The resulting table may look like the one in Figure 39.



```
                SRR10203569 SRR10203570 SRR10203571 SRR10203572 SRR10203573 SRR10203574  SRR10203575
ENSG00000000003  943.9131516 1081.681813  926.154233  928.446940  2328.75172 1936.248567 3045.9425136
ENSG00000000005    0.0000000    0.000000    0.000000    0.000000   103.40512   74.627660   42.4940651
ENSG00000000419 1073.8261263 1094.350159  755.008137 1299.561577  2052.71987 2328.383000 1446.4017622
ENSG00000000457  437.8884880  446.315559  329.190099  289.231692   213.64695  146.541587  230.1093713
ENSG00000000460  177.1540564  149.096682  110.548914  240.366064   758.01937  556.315286 1025.4699855
ENSG00000000938    0.9084823    2.923464    2.456643    2.641385     7.69129    8.141199    0.8017748
```

**Figure 38**. The first 6 features in a count table.

The above res_count table is an S4 object. You can also use the merge() function to generate the data frame object of res_count table in combination with the R function of as.data.frame(),

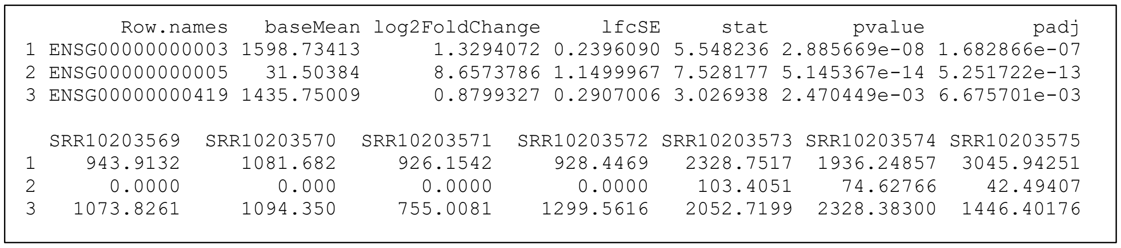
res_count <- merge(as.data.frame(res), as.data.frame(counts(dds, normalized = TRUE)), by = “row.names”, sort = FALSE)          # This generates a list mode, data frame class of data. Check using mode(res_count), class(res_count). If you use the summary() function for res, count results (count_table), and the merged data frame (res_count), you will see the differences of the data format. When practice using this tutorial, please do not copy the code from the tutorial into the RStudio because the text code may mess up your R code. For example, the double quotation marks in your Word text or PDF text may function differently in R.

To see the first 3 rows/transcripts in the res_count table, issue,

head(n=3, res_count)    # See Figure 39 for the output.

### 8.6.6 DE analyses for a sample set with multiple levels

In the case, you use the contrast argument to test just some of the samples (one of the treatments vs controls), and you do not want to extract other samples that are not tested for DE, you can use the following code, res_count <- merge(as.data.frame(res_T1_vs_C), as.data.frame(counts(dds, normalized = TRUE))[, c("T1-1", "T1-2", "T1-3", "C1", "C2', "C3")], by = "row.names", sort = FALSE) # This code uses the indexing operator [ ] to extract the normalized read counts for all three repeats of treatment 1 and that of the three repeats of control RNA-seq. The comma here indicates that we keep all the rows in the resulting table. In this case, we avoid cluttering of the table with columns of read counts for other conditions not analyzed. T here denotes "treatment", and C denotes "control". T1-1 to T1-3, and C1 to C3 are the column names.



```
        Row.names   baseMean log2FoldChange     lfcSE     stat       pvalue         padj
1 ENSG00000000003 1598.73413      1.3294072 0.2396090 5.548236 2.885669e-08 1.682866e-07
2 ENSG00000000005   31.50384      8.6573786 1.1499967 7.528177 5.145367e-14 5.251722e-13
3 ENSG00000000419 1435.75009      0.8799327 0.2907006 3.026938 2.470449e-03 6.675701e-03

  SRR10203569  SRR10203570  SRR10203571  SRR10203572 SRR10203573 SRR10203574 SRR10203575
1    943.9132     1081.682     926.1542     928.4469   2328.7517  1936.24857  3045.94251
2      0.0000        0.000       0.0000       0.0000    103.4051    74.62766    42.49407
3   1073.8261     1094.350     755.0081    1299.5616   2052.7199  2328.38300  1446.40176
```

**Figure 39**. The first three features of the res-count table, an output of command, head(n=3, res_count).

In the above code, you need to calculate the pair results first (T1 vs C). Of cause, you can do this in one step as below,

res_count <- merge(as.data.frame(results(dds, contrast = c("condition", "T1", "C")), as.data.frame(counts(dds, normalized = TRUE))[, c("T1-1", "T1-2", "T1-3", "C1", "C2', "C3")], by = "row.names", sort = FALSE) # In this code, to include the normalized read counts for the selected conditions you just specify the column names of the count table.

## Summary of Chapter 8

- This tutorial uses DESeq2 to conduct statistical analyses.
- The entire process of DE analyses can be conducted on RStudio, and HPC may not be needed. You can conduct this step on a desktop.
- The genes/rows that are not expressed in all samples can be removed before analyses.
- The result R object is generated using the DESeq2 result() function.
- The count table is generated using the DESeq2 counts() function.
- The count and statistical results can be combined using the R cbind() or merge() functions.

# Chapter 9

# Annotation and management of the result data frame

At this point, you have a table containing both statistical data and the normalized read counts for each sample (see Figure 39), but there are ENSEMBL ID (The row.names column in Figure 39) only without other information of the transcripts/genes. The tables a bioinformatician gives you include annotation information such as gene names and gene symbols. We need at least the gene symbols for the downstream analyses.

## 9.1 Convert the S4 object of the DESeq results into data frame

If you use the merge() function to generate the res_count table you can skip this step. If you use the cbind() function to produce the res_count table, you need some manipulation of the table before annotation.

The resulting tables for the DESeq() and results() functions have row names of the ENSEMBL IDs. The data set generated using the cbind() function is an S4 class object and also has row names of ENSEMBL IDs. In the annotation table, the ENSEMBL IDs constitutes a column. Before we proceed, we need to make the row names the first column of the data table. We need the R package tibble for this purpose since it has functions to manipulate row names.

Load the tibble library,

```
library(tibble)
```

Check if the results table has row names using the tibble function has_rownames(),

```
has_rownams(results(dds))

has_rownames(counts(dds, normalized = TRUE))

has_rownames(cbind(results(dds), counts(dds, normalized=TRUE)))
```

The results for the above test are FALSE, but if you check using the head() function you will know these tables have row names, which are the ENSEMBL ID.

```
head(results(dds))

head(counts(dds, normalized = TRUE))

head(cbind(results(dds), counts(dds, normalized=TRUE)))
```

We change the row names as the first column,

```
res_count <- as.data.frame(cbind(results(dds), counts(dds, normalized=TRUE)))
```

> res_count <- rownames_to_column(res_count, var = "ENSEMBL") # This code also gives the column name of "ENSEMBL" to the new column manipulated using the var argument. You can also just change the row names to a column for res or count individual objects. You need to make it a data frame first before using the function of rownames_to_column(), i.e. count <- as.data.frame(count).

Now, you can check if the first column contains the ENSEMBL IDs using head(res_count).

## 9.2 **Add annotations to the table**

Here, we use the organism-level annotation R packages. For human, the annotation package is org.Hs.eg.db. You may need to install the packages by yourself if the packages have not been installed by your HPC team.

Load the package,

> library(AnnotationDbi)
>
> library("org.Hs.eg.db") # You may see a message of: "Loading required packaged: AnnotationDbi" if you do not load AnnotationDbi first. Therefore, you can just load the library of "org.Hs.eg.db" only because it will automatically load the associated AnnotationDbi package. Please note that the quotation marks are not necessary. Note also that the H is in uppercase in the package name of org.Hs.eg.db. Otherwise, you will get a warning.

We can check how many columns of annotations we could choose from using the columns() function,

> columns(org.Hs.eg.db)

We can also find out what kind of key types we can use for annotation (usually, ENSEMBL id) using the keytypes() function,

> keytypes(org.Hs.eg.db)

We use the select() function to build up the annotation data.frame,

> anno <- AnnotationDbi::select(org.Hs.eg.db, keys = res_count$ENSEMBL, keytype = "ENSEMBL", columns = c("GENENAME", "SYMBOL")) # Please note that both "GENENAME" and "SYMBOL" are in singular forms. If you use "GENENAMES"

or "SYMBOLS", you will see a warning. Please note that the argument "keys" is in the plural form, not singular form; so is the "columns" argument; but the "keytype" argument is singular. Those make sense because you use one type of keys, but there are many keys of the selected key type; and most of the time you add >=2 columns. It will not work if you have any issue in spelling.

Now, check the dimension of the anno data frame object,

```
dim(anno)

dim(res_count)
```

You will find that the number of rows of the two objects are different. This is because one ENSEMBL Id may match multiple GENENAME and SYMBOL. We can remove the duplicates using the function duplicated(), and get the anno data frame without duplicated ENSEMBL Id using the following modified code. The code below use both the pipe operator, i.e., %>%, and the filter() function, both of which need the dplyr R package,

library(dplyr)    # This loads the dplyr package.

anno <- AnnotationDbi::select(org.Hs.eg.db, keys = count_statistic_dataframe$ENSEMBL, keytype = "ENSEMBL", columns = c("GENENAME", "SYMBOL")) %>% filter(!duplicated(ENSEMBL))    # Because both dplyr and AnnotationDbi packages use the function name of select(), we use AnnotationDbi::select() to avoid confusion. You may use select() if you do not load the package of dplyr. Please note that there are no quotation marks around ENSEMBL in the duplicated(ENSEMBL) function, and in count_statistic_dataframe$ENSEMBL. The count_statistic_dataframe is the res_count data frame. Please also note that it is "duplicated" not "duplicate".

Please note you can remove the duplicated ENSEMBL entries in a generated anno data.frame as,

anno <- filter(anno, !duplicated(ENSEMBL))    # For the usage of filter(), refer to the help information for the filter() R function.

To see the first 6 rows of the anno object,

```
head(anno)
```

The output of the above code may look like,

```
ENSEMBL                                                              GENENAME         SYMBOL
1 ENSG00000000003                                                      tetraspanin 6     TSPAN6
2 ENSG00000000005                                                      tenomodulin       TNMD
3 ENSG00000000419   dolichyl-phosphate mannosyltransferase subunit 1, catalytic         DPM1
4 ENSG00000000457                                        SCY1 like pseudokinase 3       SCYL3
5 ENSG00000000460                                 chromosome 1 open reading frame 112   C1orf112
6 ENSG00000000938                         FGR proto-oncogene, Src family tyrosine kinase   FGR
```

Now, check the dimension of the two objects again and you will find they have the same numbers of rows,

```
dim(anno)

dim(res_count)
```

Now, use the left_join() function to combine the annotations into the results data frame,

res_count__anno <- left_join(res_count, anno, by = "ENSEMBL")          # please note that it is an underscore, not a dot between "left" and "join" in the function of left_join(). Please also note that the column name for the ENSEMBL Id column should be "ENSEMBL" here. Otherwise, you will see a warning: "Error: Join columns must be present in data. x Problem with `ENSEMBL`. You need to change the name to "ENSEMBL" first. The by argument may be omitted if both data sets have the same column names of "ENSEMBL" and this is the only column that has the same column name in both tables. When the column names for the two tables are different for their ENSEMBL columns, we can define both names as identical using the by argument: by =c("Row.names" = "ENSEMBL"). Please note that when you define the two column names as the same, you need to place the two columns names in the same order as the two data frames are presented within the left_join() function. The combined table will take the first column name in the by argument as the column name of the new table.

Please note that the order of the two table objects (res_count and anno) will define the order of columns in the newly combined table generated. You can put whichever table first as you prefer. You may use the function of right_join() since the row are the same for the two tables. Please note that the different *_join() functions are ones from the package dplyr, but merge() is a base R function.

Check the first several rows of the combined table using the head() function,

```
head(res_count_anno)
```

You can briefly see the list of column names by issuing, colnames(res_count_anno)    # Or, you simply use the command names(res_count_anno) to get the same output. The colnames() and names() functions are the same.

## 9.3 Review the data frames (tables) on RStudio

You can bring about the tables using the function of View(),

View(res_count)    # Please note that unlike other R functions the "V" in the View() function is in uppercase.

View(res_count_anno)

The tables (res_count, or res_count_anno) will appear in the source editor pane. Or, you just simply click on the object name in the workspace browser pane to achieve the same as the View() function does.

## Summary of Chapter 9

- The result and count table has no annotation to each row, and annotation should be made for further downstream analyses.
- We use the R packages of AnnotationDbi and org.Hs.eg.db for annotation of human RNA-seq results.
- We use the AnnotationDbi::select() function to generate the anno object.
- We use the filter() and duplicated() functions to remove the duplicated rows in the anno object. The filter() function needs the dplyr package.
- We combine the result/count table with the anno table using the left_join() function.

# Chapter 10

# Work on the result data frame

Now, we have a result table similar to what you receive from a bioinformatician. You may want to manipulate your DE results such as sorting and extraction. Some of those manipulations can be easily done in Excel. However, R is more powerful and faster for some tasks. Here, I will show you some basic manipulation of the result table.

### 10.1 Remove rows/genes that are not expressed in all samples (both treated and untreated)

As you remember, we have removed genes before calling DESeq() for which the total read counts for all samples are less than or just 1. We may, however, have to use a higher threshold for the expressed genes, and can further reduce the size of the tables. We will use the subset() function to do this.

```
res_count_anno_expressed <- subset(res_count_anno, (NameOfcount_column1 + NameOfcount_column2 + ... + NameOfcount_columnN) > 50, names(res_count_anno))
```

This code will generate a much small data frame that lacks the genes whose total normalized read counts of all samples are less than 50. Here, you use column names (not position of the columns) to calculate the total reads. The column names can be found using the function of names() as introduced previously. Please note that there are no quotation marks around each column name. The following code extract expressed genes for  the RNA-seq samples in this tutorial with the higher threshold,

```
res_count_anno_expressed <-  subset(res_count_anno, (SRR10203569 + SRR10203570 + SRR10203571 + SRR10203572 + SRR10203573 + SRR10203574 + SRR10203575) > 50, names(res_count_anno))
```

### 10.2 Keep the genes/transcripts whose padj are < 0.05

```
res_count_anno_expressed_Q005 <- subset(res_count_anno_expressed, padj < 0.05, names(res_count_anno_expressed))        # This will generate a data frame that contains genes whose padj is less than 0.05. padj is the column name for the adjusted p values.
```

### 10.3 Keep the genes/transcripts/rows that are upregulated by 1.2 fold

```
up1dot2 <- subset(res_count_anno_expressed_Q005, log2FoldChange > log2(1.2), names(res_count_anno_expressed_Q005))        # log2FoldChange is the column name of the log2 fold change.
```

Similarly, you can keep the genes/transcripts/rows that are downregulated by 1.2 fold simply using the negative sign,

```
down1dot2 <- subset(res_count_expressed_anno_Q005, log2FoldChange < -
log2(1.2), names(res_count_anno_expressed_Q005))
```

## 10.4 Keep genes that meet multiple conditions

You can use & to subset on two or more criteria, for example, log2FoldChange and mean values at once. The logic operator & means both conditions should be met. For samples in this tutorial, the code below simultaneously select genes/rows that are downregulated at 1.2 fold at the padj < 0.05 level (meeting two conditions),

```
up1dot2_significant <- subset(res_count_anno_expressed, log2FoldChange >
log2(1.2) & padj < 0.05, names(res_count_anno_expressed))   # log2FoldChange is
the column name of the log2 fold change.
```

## 10.5 To extract all the RNA-seq data (all columns) for a list of genes

Sometimes you are interested in a list of genes and want to extract data for that set of genes from your result data frame. When your gene list is long, this is difficult to do with Excel, but it is easy in R. You can use the subset() function to achieve this as follows,

```
data4selectedGene <- subset(sourceRNAseqDataTable, ColumnNameOfGeneID %in%
vectorObjectName4GeneList, names(sourceRNAseqDataTable))
```

ColumnNameOfGeneID should be the name of the gene ID column, which could be ENSEMBL, or SYMBOL. vectorObjectName4GeneList is the object name, which define the list of genes of your interest. You need to make this vector object first. The type of gene IDs should match that in the gene ID column (ENSEML or SYMBOL) of your result table/data frame. You need the R package dplyr to use the operator of %in%.

## 10.6 Save and re-load the entire workspace

You can save your workspace in the working directory (by default) by choosing "Save Workspace As" under the Session pulldown menu of RStudio. If you want to save the workspace on the home directory, you can go to the home directory first using,

```
setwd("~/")
```

You can also use the save() function to save your workspace as a file. When using the save() function, please use the file extension name .Rdata, and use a code like this,

```
save(list=ls(), file = "myworkspace.Rdata")
```

# You need to specify the object to be saved, otherwise you will have a warning of "Warning message: In save(file = "test.Rdata") : nothing specified to be save()d. The list=ls() command defines that all the objects in the current workspace will be saved with a file name of "myworkspace.Rdata" in the current directory.

However, if you use the function of save.image(), you do not need to provide the objects,

```
save.image(file = "myworkspace.RData")
```

# save.image() is just a short-cut for 'save my current workspace'. You can simply use: save.image("myworkspace.RData"), omitting the "file =".

You can save the workspace in a specific directory by giving the path. For example,

```
save.image(file =
"/data/user/kejinhu/RNA_seq_tutorial/fastq_files/counts/RNA_seq_tutorial.RData")
```

You can also save the workspace at any step of your analyses, and come back to work on it later. After you save your workspace, you can quit RStudio using,

```
q()
```

# I am sorrty that this function should be introduced earlier for audience who has no prior R experience.

When you come back to work on your workspace, you can load the workspace by clicking the "Load Workspace" button on the Session pulldown menu. Alternatively, as introduced at the beginning of Chapter 8 you can use the load() function in the working directory which contains your file of workspace.

```
load("RNA_Seq_tutorial.RData")
```

When your workspace file is not in the working directory, you can load it by providing the path to it. For example,

```
load("/data/user/kejinhu/RNA_seq_tutorial/fastq_files/counts/RNA_seq_tutorial.RDa
ta")
```

## 10.7 Save the DE results as a csv file

You can use the write.csv() function to save individual DE result table in the csv (comma separated values) format in the working directory,

```
write.csv(res_count_anno, file = “file_name_of_your_choice.csv”, row.names = FALSE)
                    # Please do not forget the file extension of .csv.
```

If you want to save the file in a directory that is not the current working directory, you can provide the path,

```
write.csv(res_count_anno,                    file                    =
"/data/user/kejinhu/RNA_seq_tutorial/fastq_files/counts/res_count_anno.csv",
row.names = FALSE)
```

Use the list.files() function to see if the file is saved in the working directory,

```
list.files()
```

Of course, you can list the files of any directory from the working directory just by providing the path,

```
list.files("/data/user/kejinhu/RNA_seq_tutorial/fastq_files/counts/")
```

Finally, transfer data from HPC to local computer using Globus, or FileZilla as introduced in Chapter 2.

Now, you quit RStudio using q(). Please note that you quite Linux terminal using exit.

## Summary of Chapter 10

- You can save the R workspace at any steps of your DE analysis stage using the save() or save.image() functions, or simply save via the graphic menu/tabs on RStudio.
- The workspace can be re-loaded at a later time using the load() function to continue the unfinished job.
- The DE results should be saved onto your HPC as CSV files so that you can further analyze the DE results on your desktop. You can use the write.csv() function to save the file.
- You can sort or subset your DE results in RStudio more efficiently than in Excel.

# Appendix

## Software/packages used in this tutorial and their websites

SRA-Toolkit
A set of tools for management of Sequence Read Archive (SRA). for more information, visit https://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?view=toolkit_doc.

FastQC
FastQC can QC RNA-seq raw data for individual files. Website, https://www.bioinformatics.babraham.ac.uk/projects/fastqc/.

MultiQC
A tool for aggregating results of bioinfromatics based on the log files across many samples into an integrated single report. Visit, https://multiqc.info/.

nano text editor
A text editor preinstalled on many Linux distributions. It is even installed on Mac OS. The nano text editor is a simple Linux text editor and can be used to view, prepare, or edit text files, and scripts. There is a list of key combinations right in the bottom of the editor. The most useful key combination may be: CTRL+G (online help), CTRL+O (save), CTRL+X (exit), CTRL+A (to the beginning of the current line), CTRL+E (to the end of the current line), ALT+/ (to the last line of the text/file), and ALT+\ (to the first line of the text/file). Visit https://www.nano-editor.org/.

SLURM
A popular job scheduler for HPC. For detailed information (tutorial, documentations, FAQ, publications, and others), visit https://slurm.schedmd.com/ .

IGV
A tool for visualization of alignment results, https://software.broadinstitute.org/software/igv/

**Filezilla**
A platform for data transfer to and from HPC, https://filezilla-project.org/ .

**Globus**
A tool for data transfer to and from HPC, https://www.globus.org/

**Open OnDemand**

A web-based interface to HPC developed by Ohio Supercomputer Center (OSC, https://www.osc.edu/resources/online_portals/ondemand). Open OnDemand webinar: https://www.youtube.com/watch?v=OxNBSk5_sTw.

## Tools for data transfer to and from HPC

- SRA-Toolkit
- Filezilla
- Globus
- wget
- rclone
- rsync

# Linux cheat sheet for RNA-seq analyses on HPC

This section introduces some basic Linux commands used in this tutorial. It serves more like a cheat sheet not a Linux tutorial, nor a knowledge documentation. It focuses on the most commonly used command lines.

## File managements in Linux

The most critical basic skills in Linux are file managements. One basic concept in In Linux (and Unix) is that everything is a file; a file is a file; a directory (folder in Windows) is a file; a device (a hard drive, a keyboard, or a mouse) is considered to be a file; and a software exist in Linux as a file. Therefore, knowledge and skills of Linux commands about file managements are essential to work on any Linux system. The following are introductions to some basic commands of file management.

### What are inside the room? - List contents of a directory

ls is one of the mostly used Linux command. It prints out the contents of a directory on your terminal screen. After login to HPC (or on your desktop terminal), simply issue,

```
ls
```

This will list the names of files in the current directory by default. A Linux command takes options, and the frequently used option of ls is -l, which allows ls to output files in the long format. The long format provides more information about each file and may have the following appearance,

```
total 2576
-rw-rw-r--   1  kejinhu  kejinhu   827   Dec 27 15:48   STAR_Index.slurm
drwxrwxr-x  12  kejinhu  kejinhu   8192 Jan  2 10:58   fastq_files
drwxrwxr-x   3  kejinhu  kejinhu   4096  Dec 28 10:18   humanGenomeIndex
drwxrwxr-x   2  kejinhu  kejinhu   4096  Dec 30 12:42   multiqc_data
-rw-rw-r--   1  kejinhu  kejinhu   394   Dec 31 13:26   samtools_index.slurm
-rw-rw-r--   1  kejinhu  kejinhu   472   Dec 30 14:58   samtools_sort.slurm
```

Another commonly used option to ls is -h, which means outputing the files in human readable format. Try,

```
ls -l -h
```

Multiple potions can be combined. For example, the script of ls -l -h can be used as,

```
ls -lh
```

There are hidden files in any directory. A hidden file starts with a dot. There are at least two hidden file, "." and "..". Single dot means the current directory, and double dots indicate parent directory (the directory one level up). Therefore, ls . functions the same as ls, and ls .. prints the files inside the parent directory. We need the option of -a to see the hidden files. Try,

```
ls -a
```

The three options of -a -l and -h can be combined,

```
ls -ahl        # This functions the same as ls -a -l -h does.
```

When the options are combined the order of the options does not matter and ls -ahl, ls -hal and ls -lha do the same thing.

## Looking for help

A single letter option of a Linux command is indicated by a single dash -, but if you use a multiple-letter option (i.e., word option) two dashes are used, i.e. --. The most useful option for all Linux command is --help. The --help option outputs the help page of a Linux command line. For example,

```
ls --help      # This displays the help page for the command of ls.
```

Please note that Linux has a help command, which is used differently from the --help option. Please try,

```
help help      # This prints out the help page of the help command. The –help
option is more useful than the help command. Please try you self, help cd; help echo;
help pwd.
```

Behind each Linux command is a software, and you can find the versions of many commands using the --version option of a command. For example,

```
ls --version
```

The information outputted by the above script indicates that ls is one of the GNU core utilities.

Another way to find out help information is using the manual command, man, try,

```
man ls
```

The above code will bring about the manual pages on the terminal screen. The manual is generally longer than a single screen can accommodate, and you can navigate along the manual line by line using the down or up arrow keys, or page by page using the page-up or page-down keys (or key combinations). To quit the manual pages and return to the shell terminal, you just hit the letter *q* key on your keyboard.

Of course, you can google any of your Linux questions and you can always find a solution to your questions. I do googling all the time whenever I have a Linux issue.

### Make a storage room using the command of mkdir

When you login to your HPC account, you will land on your home directory, i.e., /home/yourName. You can make your own directory to hold your files using the command of mkdir, which means "make directory". mkdir is also a GNU core facility (You can see this by issuing mkdir --version). The syntax is *mkdir directory_name*, for example,

```
mkdir test
```

The above code generates a directory named *test*. Issue ls or ls -l to see the generated directory of test.

You can make multiple directories in one code. Within the test directory, make subdirectory of example1, example2 and example3,

```
mkdir example1 example2 example3
```

Use ls or ls -l to see your newly generated directory.

### From one room to another – the hange directory command cd

The second essential and most used command is cd, i.e., Change Directory. You can go to the new directory *test* you just generated,

```
cd test
```

Now, use ls or ls -l to see the contents of the *test* directory, and you will know there is none. Make sub-directory within the test directory,

```
mkdir test1 test2 test3
```

use ls or ls -l to see the newly generated subdirectories. You will see you have generated three sub-directories.

As described above the double dots denote parent directory. Therefore, you can go to the immediate parent directory using the code below,

```
cd ..
```

You are back in the home directory in this case. You can go directly to any deep branch directory by providing the path to it. For example,

```
cd test/test1
```

The tilde sign ~ denotes the home directory, you can go to home directory from any directory using the code,

```
cd ~
```

The above code navigates you from the test1 directory to the home directory, and now you can see a tilde sign after your username.

You can go to the last directory you were using,

```
cd -
```

### Where am I?

Another essential Linux command line is for printing out the path to the working directory, i.e., pwd,

```
pwd     # This outputs the full path to the current directory.
```

## Useful Linux techniques

### Copy and paste the text

To copy target text, you can just highlight the text to be copied using your mouse cursor. When you release your mouse after choosing the text, you will see a flashing scissors. This indicate that the text is copied onto the clipboard, and you can now paste it as needed. For iMAC, you can use the key combination of “command + V” to paste the copied text from the clipboard to the terminal.

### Reuse previously executed command lines

You can use the up-arrow key to bring back the executed commands. You can keep hitting the up-arrow key till you reach the command you are interested in, and you can hit enter to re-execute the same command, or you can modify your script and execute the revised script. The down-arrow key can be used similarly.

### Path expansion for file/directory management

The wildcard * is the most used path expansion character. * can be used to match any character or no character in a character string. It is useful to manipulate the file names in writing codes for RNA-seq analyses of multiple files. Other means of path expansions include "?" (question mark), "[ ]" (square brackets) and "{ }" (curly braces). ? matches exactly one character; [ ] are used to specify characters for which one of them will be matched; { } are used to list character string of patterns to be matched.

### Bash comments

The pound (or hash) sign # is the comment indicator in Linux. Within script, anything after # till the end of the line is a comment and is ignored by the bash interpreter. In this tutorial, # is also used to indicate comments after a command line/script.

## The most useful key skills in Linux

q key, the q key means quit from the online manual page or an opened text file.

tab key, the tab key functions as a completion key in both Linux terminal and R. This key will save you a lot of time and avoid typos.
Control + C, pressing the Control and C keys together will terminate a process in Linux.
Control + D, exit the Linux terminal. It is equivalent to exit followed by ENTER key.
Control + E, take the cursor to the end of the command line.
Control + A, go to the beginning of the command line in the screen.

## List of command Linux commands, operators used in this tutorial

cd, change directory

cp, copy files

cat, concatenate files and print on the standard output echo, output the line of text/string, which is passed as an argument.

gunzip, compress or uncompress files.

ls, list the files in the current directory. The mostly used option is -l, and you can combine the -h with the -l options, i.e., -hl to display the more human readable output.

lscpu, display information about the cpu architecture.

pwd, print out the path of the current directory.

mkdir, make directory.

nano, open or create a text file, and then view, and/or edit it in Linux environment.

rm, rm files and/or directories.

man, an interface for the online reference manuals. It is very useful whenever you need help about a Linux command. Examples: you can try to retrieve the manuals for cp, or rm, or pty: *man cp*; *man rm*; *man pty*.

help, a command to get the help page of Linux command. The syntax is help command-line. For example, help cd; help help; help pwd. Please note that Linux command lines have a --help option, which has a different syntax (see below).

--help, a command option that retrieve online help pages when used with a command. Example, try: cp --help.

less, displays the content of a file one page first, and allows for subsequent navigation within the file page by page or line by line using the arrow keys and page keys.

head, output the first part of a file.

mv, move files around, and can be used to re-name files.

module spider, list the specified module.

module load, load the specified module.

nano, open a text file in the nano text editor.

rclone copy, copy files and directories between HPC and cloud storage.
rclone config, configurate access to your cloud storage.

srun, run a parallel job on cluster managed by SLURM. Call exit to exit the interactive srun mode.

sbatch, submit a non-interactive batch script to SLURM.

squeue, view information about jobs managed by SLURM. The mostly used option with the squeue command is the user option -u, for which the value should be your HPC username. It will print out a long list of jobs if you do not use any option, i.e., squeue.

scancel, cancel batch jobs.

rsync, a utility that can transfer data (files and directory) between your local computer and HPC.

Output redirection operator >, > diverts the standard output of a command into the specified file.

The pipe operator |. In Linux, the pipe operator is used to redirect the output of one command to another command to process as input data. The syntax is, *command-1 | command-2 | ..... command-N*.

Line continuation operator \, the backslash serves as a line continuation operator when placed in the last place of a script line. Backslash is frequently used in RNA-seq coding since you encounter long command line a lot when preparing code for RNA-seq analyses. Please note that \ is the last character and there is even no white space after \.

tail, output the last part of a file.

scontrol show job [job ID], this will print out the basic information about the listed job (identified by the job ID) including number of CPUs, tasks, nodes, CPU/task, memory and others.

scontrol show partition. This will print out the partition systems on your HPC. Scontrol.

scontrol show node [node ID], this will output the information of the specified node.

wget, the non-interactive network downloader.

wc, count the number of bytes, lines, words, characters, and others of specified files, and print out the count results.

The for loop is frequently used in RNA-seq analysis to analyze multiple samples using the same code. Linux for loop has the syntax of, for variable in string; do application code; done. In this syntax, the red texts are invariable; the purple ones are user defined. In RNA-seq, the string is usually a set of file names; the variable represents the file names iterating over the string list; the application code is package-specific instruction of compute job.

## Major R packages used

- DESeq2
- tibble
- AnnotationDbi
- org.Hs.eg.db
- dplyr

## References

1. Shotts W. The Linux Command Line.
2. The-Standish-Group. Modernization - Clearing a pathway to success. Available at: https://www.standishgroup.com/sample_research_files/Modernization.pdf.
3. Dobin A, Davis CA, Schlesinger F, et al. STAR: ultrafast universal RNA-seq aligner. Bioinformatics. 2013;29:15-21.
4. Dobin A, Gingeras TR. Mapping RNA-seq Reads with STAR. Curr Protoc Bioinformatics. 2015;51:11 14 11-11 14 19.
5. Sterling T, Anderson M, Brodowicz M. High performance computing : modern systems and practices. Cambridge, MA: Morgan Kaufmann; 2018: xxviii, 689 pages.
6. Hu K. Become Competent in Generating RNA-Seq Heat Maps in One Day for Novices Without Prior R Experience. Methods Mol Biol. 2021;2239:269-303.
7. Hu KJ. Become Competent within One Day in Generating Boxplots and Violin Plots for a Novice without Prior R Experience. Method Protocol. 2020;3.
8. Martin T. The Undergraduate Guide to R.
9. Venables WN, Smith DM, Team RC. An Introduction to R.
10. Chalker A, Franz E, Rodgers M, et al. Open OnDemand: State of the platform, project, and the future. Concurr Comp-Pract E. 2021;33.
11. Hu K, Ianov L, Crossman D. Profiling and quantification of pluripotency reprogramming reveal that WNT pathways and cell morphology have to be reprogramed extensively. Heliyon. 2020;6:e04035.
12. Alnasir JJ. Fifteen quick tips for success with HPC, i.e., responsibly BASHing that Linux cluster. PLoS Comput Biol. 2021;17:e1009207.
13. Sterling T, Anderson M, Maciej B. The Essential Resource Management. High Performance Computing - Modern Systems and Practices. Cambridge, MA, USA: Morgan Kaufmann - Elsevier; 2018:141-190.
14. SchedMD. SLURM Wordload Manager. Available at: https://slurm.schedmd.com/.
15. Li H, Handsaker B, Wysoker A, et al. The Sequence Alignment/Map format and SAMtools. Bioinformatics. 2009;25:2078-2079.
16. IGV. Integrative Genomics Viewer. Available at: https://software.broadinstitute.org/software/igv/home.
17. Thorvaldsdottir H, Robinson JT, Mesirov JP. Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration. Brief Bioinform. 2013;14:178-192.
18. Robinson JT, Thorvaldsdottir H, Winckler W, et al. Integrative genomics viewer. Nat Biotechnol. 2011;29:24-26.