

Analyzing sequencing data in the cloud*

Class 17: Obtaining and processing SRA datasets on AWS

Barry Grant

2023-03-01

Table of contents

Background	2
Accessing the AWS console	2
Re-starting a previous instance	3
Starting a new instance	3
Connecting to your instance	4
Simplifying SSH and SCP with environment variables	4
The Sequence Read Archive (SRA)	7
The SRA-toolkit package	8
A mini example: from a given paper to raw reads	10
Working with RNA-Seq data	14
Transcript quantification via pseudoalignment	16
Building a transcript index	17
Quantifying transcripts	18
Scripting repetitive tasks	19
Transfer your results	20
IMPORTANT: Stop your instance!	20
Downstream analysis	21
Principal Component Analysis	23
OPTIONAL: Differential-expression analysis	25

*<http://thegrantlab.org/teaching/>

Background

In bioinformatics we often need to analyze data-sets that are too large (or would take too long) to analyze on our local lab computers. The goal of this hands-on session is to show you how to obtain and work with large biomolecular sequencing data-sets using **cloud computing** resources. In particular, we will query, download, decompress and analyze data from NCBI's main **Sequence Read Archive** (SRA) the world's largest publicly available repository of high throughput sequencing data.

What is cloud computing?

Cloud computing allows access to arbitrary amounts of compute resources that are physically located elsewhere. Typically you pay for what you use rather than having to pay upfront for new hardware and all its associated maintenance costs. Cloud computing resources exist on servers managed by cloud providers. The most popular cloud providers include Amazon (**Amazon Web Services**, a.k.a. **AWS**), Google (**Google Compute Engine**) and Microsoft (**Azure**) (Figure 1). For academic work in the US we also have NSF/XSEDE (who manage access to JetStream and other supercomputers around the country).

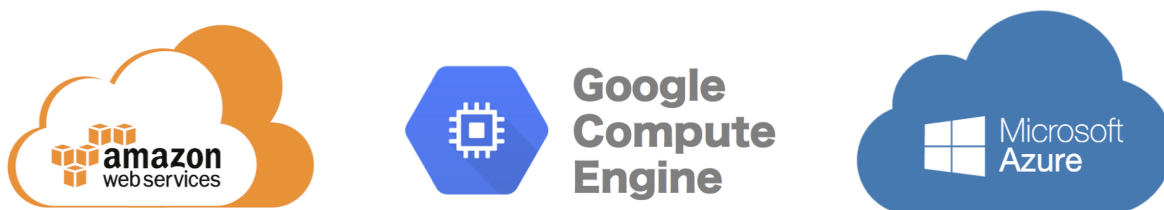


Figure 1: Major cloud compute providers. At the time of writing Amazon's **AWS** is the market leader by a rather wide margin.

In our [last hands-on session](#) we introduced important cloud concepts and walked through configuring and launching **AWS EC2 instances**.

Accessing the AWS console

The AWS console is a password protected website where you can configure, launch and control your EC2 instances. We introduced this console last day and we will not re-cover that material here beyond the instructions for starting an instance.

If you are an enrolled student in this course you can access your own AWS console at <https://awsed.ucsd.edu/> This will ask for your regular UCSD single-sign-on details and then you should be able to select our course and be re-directed to the AWS console (Figure 2):

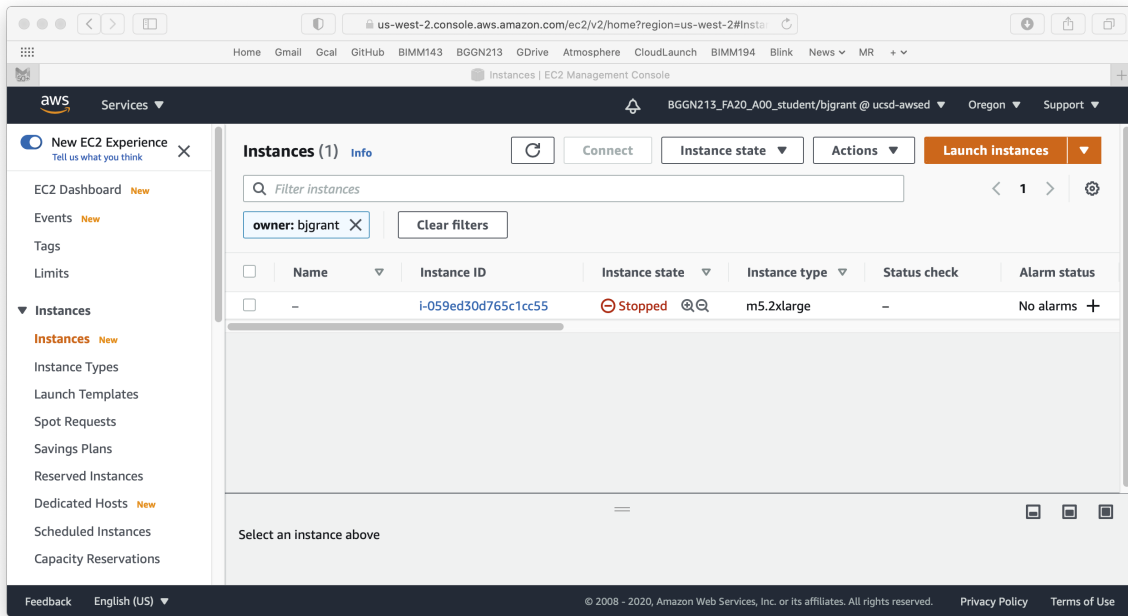


Figure 2: AWS console showing a stopped EC2 instance and buttons to change instance state i.e. start/stop and launch a new instance.

Re-starting a previous instance

Note that in the image above I have a **stopped** instance listed. This is the one I used for our last class. We can re-start this instance for use today. To do this select your instance (by clicking on the radio button next to the stopped instance name). Then click “**Instance State**” > “**Start Instance**”. Note that the instance will have a new IP address that we can find from selecting instance and then clicking “**Connect**” > **SSH client**. See the [connecting to your instance](#) section below.

Starting a new instance

If you don’t have a previous instance listed you can start a new one by following the same steps from our last class. Briefly, click the orange **Launch instances** button in the upper left and on the resulting launch page:

- Give a useful **name** for your instance (e.g. a combination of class number and your UCSD username, for example: `class17_bjgrant`);
- Select **Ubuntu** as the OS and the version **Ubuntu Server 22.04**

- Instance type **m5.2xlarge**
- You can use your previous **key pair** from the last class or (if in doubt about where to find it) *create a new one* and use it.
- Select **existing security group** and chose **BIMM143/BGGN213 FA20**
- Configure storage with 100Gib
- Verify all your settings in the right hand panel and click “**Launch instance**”.

Connecting to your instance

After configuring and launching your instance it will take a short amount of time to setup and become available for use.

Once available you can login via **SSH** in your Terminal as we did previously.

- Recall that you need to provide your **keyfile** along with your new instance IP address. See the “**Connect to instance**” link and “SSH client” link for instructions and details of your specific instance IP address (e.g. the boxed text in Figure 3 example image below):
- You can use your favorite Terminal application or the RStudio Terminal to enter the appropriate SSH command. For example, in the below image I am using my keyfile `-i bioinf_barry.pem` along with my username `ubuntu` and the EC2 address provided from the AWS console (Figure 4):

Note that if it is your first time connecting to your instance it will ask if you are sure you want to connect, to which you can answer **yes**.

Now that we have successfully gained access to a UNIX based machine with sufficient memory and cpu resources we can introduce the main Sequence Read Archive - currently the largest publicly available repository of high throughput sequencing data.

Simplifying SSH and SCP with environment variables

The SSH and SCP commands for connecting to and copying from your AWS instance can get long and error-prone — a single typo in the key filename or IP address will cause the connection to fail.

One clean solution is to store these values in **shell environment variables** so that you only have to type (and check) them once.

- On your **local computer**, set a variable pointing to your key file. For example, my keyfile is called `bioinf_barry.pem` and it lives in my Downloads folder (i.e. `~/Downloads`):

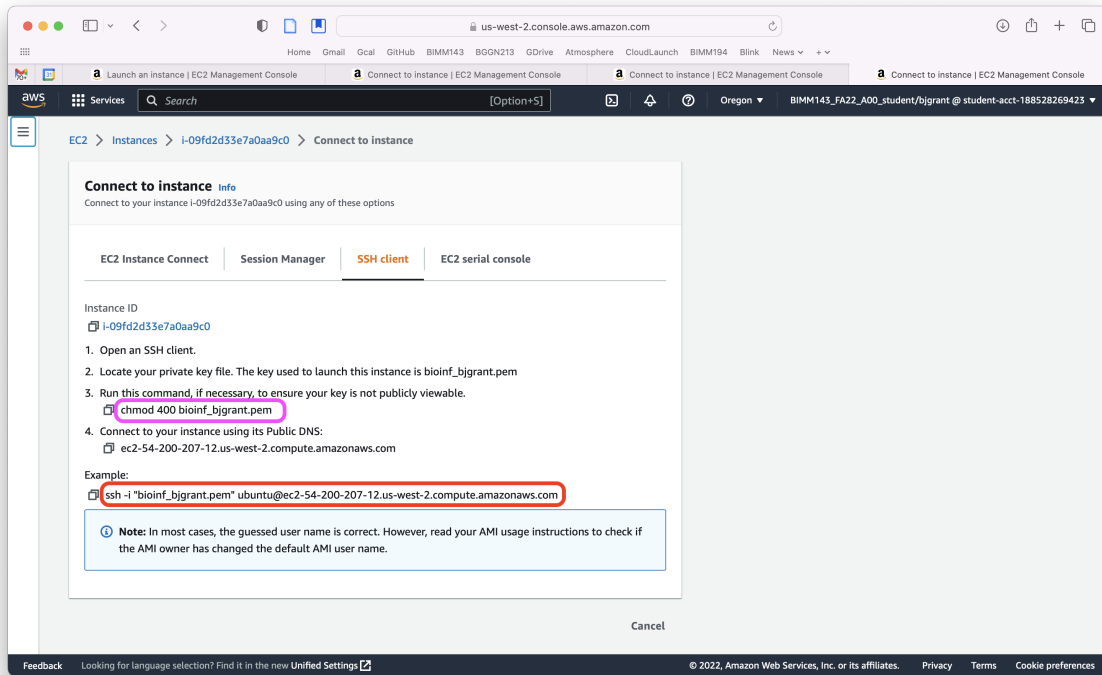


Figure 3: AWS Connect to instance dialog showing the SSH client tab with the example ssh command and IP address highlighted

```
ubuntu@ip-172-31-49-53: ~ (ssh)
Downloads (-bash)
blitz-2:~> cd Downloads/
blitz-2:Downloads> chmod 400 bioinf_barry.pem
blitz-2:Downloads> ssh -i bioinf_barry.pem ubuntu@ec2-44-234-27-254.us-west-2.compute.amazonaws.com
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-1029-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Mon Dec  7 04:33:00 UTC 2020

System load:  0.0          Processes:           147
Usage of /:   16.8% of 7.69GB   Users logged in:    0
Memory usage: 0%          IPv4 address for ens5: 172.31.49.53
Swap usage:   0%

1 update can be installed immediately.
0 of these updates are security updates.
To see these additional updates run: apt list --upgradable

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-49-53:~$
```

Figure 4: Terminal window showing a successful SSH login to an EC2 instance

Listing 1 On YOUR computer

```
export KEY=~/.Downloads/bioinf_barry.pem
```

- Now set a variable for your instance address (**N.B.:** copy YOUR specific address from the AWS console):

Listing 2 On YOUR computer

```
export SERVER=ubuntu@ec2-34-210-54-74.us-west-2.compute.amazonaws.com
```

- You can verify both variables are set correctly with `echo`:

Listing 3 On YOUR computer

```
echo $KEY  
echo $SERVER
```

Now you can use these variables in place of the full strings for all subsequent `ssh` and `scp` commands, making them much shorter and less error-prone:

Note: These variables only persist for the duration of your current Terminal session. If you close your Terminal and reopen it (or open a new Terminal), you will need to re-run the two `export` commands above. *I suggest you make a note of these in a reference text file on your computer.*

The Sequence Read Archive (SRA)

The **Sequence Read Archive** (SRA), <https://www.ncbi.nlm.nih.gov/sra> is NCBI's major repository for sequencing data and includes all types of high-throughput sequencing experiments.

Many other domain specific databases will link to entries in the SRA. For example the **Gene Expression Omnibus** (GEO) that focuses on gene expression data links to SRA for retrieval of raw data from sequencing-based experiments whose processed data is described in a *GEO record*. To get the actual raw data you need to go to the associated SRA entry.

Sequencing data stored on SRA is highly-compressed to enable efficient storage and downloading. The main consequences of this, beyond faster download speeds, is that *we must use special tools to decompress these records into usable FASTQ files*.

Listing 4 On YOUR computer

```
# Connect to your instance
ssh -i $KEY $SERVER

# Transfer results.tsv back to your current directory
scp -r -i $KEY $SERVER:~/results.tsv .
```

The SRA-toolkit package

The **SRA-toolkit** software package provides functionality to retrieve data from the SRA, and to extract the retrieved records to FASTQ files.

Instructions for obtaining the package on different platforms can be found on the projects [GitHub repo](#). Note that precompiled binaries are available for Ubuntu that we will make use of here.

- On your AWS instance download the Ubuntu binaries with the `curl` command. Then `unzip` and `untar` to be able to use the software:

Listing 5 On your AWS instance

```
# Download
curl -O https://ftp-trace.ncbi.nlm.nih.gov/sra/sdk/current/sratoolkit.current-ubuntu64.tar.gz

# Unzip and Untar
tar -zxvf sratoolkit.current-ubuntu64.tar.gz
```

Note the name of the directory that tar created. The name of this directory *changes with each release* and varies by platform, i.e. it follows the pattern `sratoolkit.<release>-<platform>` e.g. at the time of writing a directory called `sratoolkit.3.3.0-ubuntu64` was produced.

There should be a new directory called `sratoolkit.3.3.0-ubuntu64` (or a similar name with a higher number as the version changes in the future). Check for this directory with your regular `ls` command. Then `cd` into the associated `bin/` directory, (e.g. Figure 5)

In the `bin/` directory there are lots of programs that we can use to work with SRA data. We will focus on two main programs - **prefetch** and **fasterq-dump**.

The first of these, **prefetch**, retrieves compressed data from SRA. The second, **fasterq-dump**, reconstructs the actual FASTQ file(s) from the output of `prefetch` that we want to analyze.

```
ubuntu@ip-172-31-9-75: ~/sratoolkit.3.0.1-ubuntu64/bin
ubuntu@ip-172-31-9-75:~$ ls
sratoolkit.3.0.1-ubuntu64  sratoolkit.current-ubuntu64.tar
ubuntu@ip-172-31-9-75:~$ clear
ubuntu@ip-172-31-9-75:~$ ls
sratoolkit.3.0.1-ubuntu64  sratoolkit.current-ubuntu64.tar
ubuntu@ip-172-31-9-75:~$ cd sratoolkit.3.0.1-ubuntu64/bin/
ubuntu@ip-172-31-9-75:~/sratoolkit.3.0.1-ubuntu64/bin$ pwd
/home/ubuntu/sratoolkit.3.0.1-ubuntu64/bin
ubuntu@ip-172-31-9-75:~/sratoolkit.3.0.1-ubuntu64/bin$ ls
abi-dump          kar                sra-sort.3
abi-dump.3        kar.3              sra-sort.3.0.1
abi-dump.3.0.1    kar.3.0.1          sra-stat
abi-load          kdbmeta            sra-stat.3
abi-load.3        kdbmeta.3          sra-stat.3.0.1
abi-load.3.0.1    kdbmeta.3.0.1     srapath
align-info        latf-load           srapath-orig.3.0.1
align-info.3      latf-load.3         srapath.3
align-info.3.0.1  latf-load.3.0.1    srapath.3.0.1
bam-load          ncbi                sratools.3.0.1
bam-load.3        pacbio-load         srf-load
bam-load.3.0.1    pacbio-load.3       srf-load.3
cache-mgr         pacbio-load.3.0.1  srf-load.3.0.1
cache-mgr.3       prefetch            test-sra
cache-mgr.3.0.1   prefetch-orig.3.0.1 test-sra.3
cg-load           prefetch.3          test-sra.3.0.1
```

Figure 5: Terminal showing the sratoolkit programs

Listing 6 On AWS

```
ls
cd sratoolkit.3.3.0-ubuntu64/bin
pwd
```

We can check if these programs are working by running them with the `--version` or `--help` options.

For example:

Listing 7 On AWS

```
~/sratoolkit.3.3.0-ubuntu64/bin/prefetch --version
#/home/ubuntu/sratoolkit.3.3.0-ubuntu64/bin/prefetch : 3.0.1
```

Note that even though I can use the TAB key to help it is still cumbersome to type the full PATH to the `prefetch` program every time we want to run it. To avoid needing to do this we can add its location to a special `$PATH` environment variable:

Listing 8 On AWS

```
export PATH=$PATH:/home/ubuntu/sratoolkit.3.3.0-ubuntu64/bin
```

Now we will not need to type the absolute PATH every time we want to run one of the sra-tools programs from its `bin/` directory, e.g. this will now just work

With this setup complete we are now ready to get some sequencing data to work with.

A mini example: from a given paper to raw reads

Let's say we wanted to obtain the ChIP-seq data from an ovarian cancer DNA sample, such as that cited in:

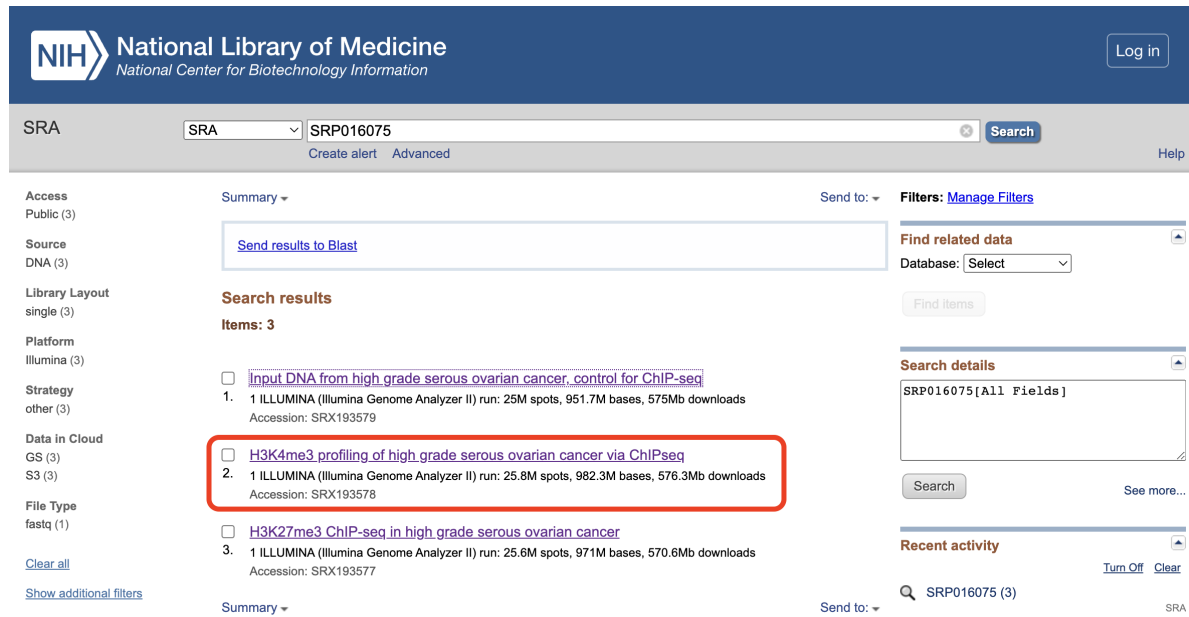
N Chapman-Rothe et al, "[Chromatin H3K27me3/H3K4me3 histone marks define gene sets in high-grade serous ovarian cancer that distinguish malignant, tumour-sustaining and chemo-resistant ovarian tumour cells](#)" *Oncogene* 32(38):4586 (2013).

Visit the full text link for this manuscript from the link above. Note that the manuscript lists an **SRA accession** for the project of '*SRP016075*' under the **Accession codes** section.

Listing 9 On AWS

```
prefetch --version  
  
#prefetch : 3.3.0
```

Searching for the corresponding record on the SRA database, we will [find a list of the sequencing runs](#) associated with this manuscript (Figure 6).



The screenshot shows the NCBI SRA search results page for accession SRP016075. The page header includes the NIH logo and the text "National Library of Medicine National Center for Biotechnology Information". The search bar contains "SRP016075" and a "Search" button. The results are displayed in a list format with three items. The second item, "H3K4me3 profiling of high grade serous ovarian cancer via ChIPseq", is highlighted with a red box. The page also includes a sidebar with navigation links, a "Send to" dropdown, and a "Filters" section.

Figure 6: NCBI SRA search results page for accession SRP016075

Specifically look at the ChIP-seq with the H3K4me3 antibody, which can be found at: <https://www.ncbi.nlm.nih.gov/sra/SRX193578>. See Figure 7:

From the corresponding web page (Figure 7 above), there is a run identifier listed: 'SRR600956'. This is the SRA identifier we need to obtain the raw sequencing data with the `prefetch` program.

To download the corresponding data, call the `prefetch` program (using either its absolute path `~/sratoolkit.3.3.0-ubuntu64/bin/` or if you have followed the instructions above simply its name) with the run accession number as its only argument:

Run an `ls` to see the resulting directory - note that the contents here is compressed and not yet usable for further analysis.

SRX193578: H3K4me3 profiling of high grade serous ovarian cancer via ChIPseq
 1 ILLUMINA (Illumina Genome Analyzer II) run: 25.8M spots, 982.3M bases, 576.3Mb downloads

Submitted by: IMPERIAL COLLEGE LONDON

Study: Homo sapiens Epigenomics
[PRJNA177332](#) • [SRP016075](#) • [All experiments](#) • [All runs](#)
[show Abstract](#)

Sample: Primary high grade serous ovarian tumour used for histone modification profiling via ChIP-seq
[SAMN01761041](#) • [SRS367927](#) • [All experiments](#) • [All runs](#)
 Organism: [Homo sapiens](#)

Library:
 Instrument: Illumina Genome Analyzer II
 Strategy: CHIP-Seq
 Source: GENOMIC
 Selection: ChIP
 Layout: SINGLE

Spot descriptor:
 1 forward

Runs: 1 run, 25.8M spots, 982.3M bases, [576.3Mb](#)

Run	# of Spots	# of Bases	Size	Published
SRR600956	25,849,655	982.3M	576.3Mb	2015-07-22

Figure 7: NCBI SRA page for experiment SRX193578

To obtain usable raw data we first need to reconstruct the FASTQ file(s) with the **fasterq-dump** program:

This will take some time but eventually result in a FASTQ file, appearing in your current working directory, which constitutes the raw data from the corresponding sequencing run.

Q. What shell command can you use to view the top few files of your FASTQ file?

Q. What length are these sequence reads?

Q. Can you use the **grep** command to determine how many total reads are in this file?

Q. Does your number of reads from **grep** match the name of the last read in the file? If not why not?

Tip

You can use the **head** command to see the start of the resulting FASTQ file and **tail** to see the end.

Listing 10 On AWS

```
cd
pwd
#/home/ubuntu

prefetch SRR600956

#2023-02-27T20:38:08 prefetch.3.0.1: Current preference is set to retrieve SRA Normalized Format
#2023-02-27T20:38:08 prefetch.3.0.1: 1) Downloading 'SRR600956'...
#2023-02-27T20:38:08 prefetch.3.0.1: SRA Normalized Format file is being retrieved, if this is
#2023-02-27T20:38:08 prefetch.3.0.1: Downloading via HTTPS...
#2023-02-27T20:38:27 prefetch.3.0.1: HTTPS download succeed
#2023-02-27T20:38:28 prefetch.3.0.1: 'SRR600956' is valid
#2023-02-27T20:38:28 prefetch.3.0.1: 1) 'SRR600956' was downloaded successfully
```

Listing 11 On AWS

```
fasterq-dump SRR600956
```

Listing 12 On AWS

```
head SRR600956.fastq

#@SRR600956.1 HWI-EAS486_0002:3:1:1382:1342 length=38
#GTGTTCCAAATGCTGCAAATGGGTGCAATGTATGTTA
#+SRR600956.1 HWI-EAS486_0002:3:1:1382:1342 length=38
#D?BCCA?BDBDBACD@=??BAAC>CBBBBBCBBBD?%%
#@SRR600956.2 HWI-EAS486_0002:3:1:1382:5487 length=38
#GATGATAGTTTCTTTTGCCGTTAGCACAAATTTTCCAA
#+SRR600956.2 HWI-EAS486_0002:3:1:1382:5487 length=38
#DCEECEAECEFFDFECEEEFFFFDB?BBADEEEEEE???
#@SRR600956.3 HWI-EAS486_0002:3:1:1382:4694 length=38
#TGTAGGCTCCACCTCTGGGGCAGGGCACAGACAAACA
```

Note that each new read starts with the pattern `@SRR600956`. You can use `grep` with this pattern to determine how many total reads are in this file:

Listing 13 On AWS

```
grep -c "@SRR600956" SRR600956.fastq  
  
#25849655
```

Working with RNA-Seq data

In this section we will move on to an arguably more interesting RNA-Seq dataset from a human cancer cell line: two replicates of a cell line treated with CRISPR-Cas9 targeting a gene of interest, and two replicates of the cell line treated with a control construct as described in:

Zhang et al. “[Identification of focally amplified lineage-specific super-enhancers in human epithelial cancers](#)”. Nat Genet 2016 Feb;48(2):176-82. PMID: 26656844

We will start with their SRA accession numbers, download their raw data and follow up with a complete analysis from raw reads to gene specific counts, principal component analysis and differential expression analysis.

The GEO entry for the study in question can be found at <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE72001>

Let’s use the SRA toolkit to download the study’s data and extract it into FASTQ format:

Side-note: With single-end reads we will have only one file for each sample, with paired-end reads (which are more common in RNA-seq) we will have two files for each sample as we have here.

Listing 14 On AWS

```
prefetch SRR2156848  
  
#2023-02-27T20:47:07 prefetch.3.0.1: Current preference is set to retrieve SRA Normalized Format  
#2023-02-27T20:47:07 prefetch.3.0.1: 1) Downloading 'SRR2156848'...  
#2023-02-27T20:47:07 prefetch.3.0.1: SRA Normalized Format file is being retrieved, if this :  
#2023-02-27T20:47:07 prefetch.3.0.1: Downloading via HTTPS...  
#2023-02-27T20:47:14 prefetch.3.0.1: HTTPS download succeed  
#2023-02-27T20:47:15 prefetch.3.0.1: 'SRR2156848' is valid  
#2023-02-27T20:47:15 prefetch.3.0.1: 1) 'SRR2156848' was downloaded successfully  
#2023-02-27T20:47:15 prefetch.3.0.1: 'SRR2156848' has 0 unresolved dependencies
```

Recall that the `prefetch` program prepares temporary files for obtaining sequence data efficiently from the SRA. With this run, you can use `fasterq-dump` to extract the data in FASTQ format to a file in the current directory.

Listing 15 On AWS

```
fasterq-dump SRR2156848

#Read 2959900 spots for SRR2156848
#Written 2959900 spots for SRR2156848
```

Note that as this sequencing run is a paired-end library, this command results in two separate files for the different mate-pairs. Now if we look at the contents of the current directory, we should see the following:

Listing 16 On AWS

```
ls

#SRR2156848/
#SRR2156848_1.fastq
#SRR2156848_2.fastq
```

Q. How would you check that these files with extension `.fastq` actually look like what we expect for a FASTQ file? You could try printing the first few lines to the shell standard output:

Listing 17 On AWS

```
# Fill in the blank!
___ SRR2156848 1.fastq
```

Q. How could you check the number of sequences in each file?

Listing 18 On AWS

```
# Fill in the blanks
---- -___ "___" SRR2156848_1.fastq
2959900
```

Q. Check your answer with the bottom of the file using the `tail` command and also check the matching mate pair FASTQ file. Do these numbers match? If so why or why not?

Q. Download the other 3 datasets (SRR2156849, SRR2156850 and SRR2156851) we need for our analysis, first with `prefetch` and then process with `fasterq-dump`

Tip

Listing 19 On AWS

```
prefetch SRR2156849 SRR2156850 SRR2156851
fasterq-dump SRR2156849 SRR2156850 SRR2156851
```

Q. Check you have pairs of FASTQ files for all four datasets and that they have the same number of counts in each pair?

Listing 20 On AWS

```
# fill in the blank
___ *.fastq
grep -c "_____" *.fastq
```

Transcript quantification via pseudoalignment

A fast approach to analysis of RNA-seq data makes use of the fact that if you only wish to quantify the expression of a gene, you don't need to know where in the gene the read maps to. In fact, it is enough to know which genes a read might have come from. This is the principle behind **Kallisto**. In brief, Kallisto compares each RNA-seq read against an indexed table of transcript sequences to find the sets of transcripts that each read could have come from. These sets are then used to estimate the most likely set of transcript counts over the whole RNA-seq library.

- To begin with we need to install kallisto software on our AWS instance:

We can check the program works by calling the `kallisto` executable with its full PATH:

Listing 21 On AWS

```
wget https://github.com/pachterlab/kallisto/releases/download/v0.44.0/kallisto_linux-v0.44.0

# Unzip and Untar the resulting file
___ -___ kallisto_linux-v0.44.0.tar.gz
```

Listing 22 On AWS

```
~/kallisto_linux-v0.44.0/kallisto

#kallisto 0.44.0
#
#Usage: kallisto <CMD> [arguments] ..
#
#Where <CMD> can be one of:
#
#  index      Builds a kallisto index
#  quant      Runs the quantification algorithm
#  pseudo     Runs the pseudoalignment step
#  h5dump     Converts HDF5-formatted results to plaintext
#  inspect    Inspects and gives information about an index
#  version    Prints version information
#  cite       Prints citation information
#
#Running kallisto <CMD> without arguments prints usage information for <CMD>
```

- For ease of use and to avoid having to type the full PATH each time we want to use the program we can add its location to our \$PATH environment variable as we did previously for the **sra-tools** package:

Now we should be able to run it by simply calling **kallisto**.

Q. Can you run **kallisto** to print out its citation information?

Building a transcript index

Let's pull the **hg19 Ensembl** reference transcriptome directly from the ENSEMBLE ftp site:

- We can use the **wget** command or **curl** command for this.

Listing 23 On AWS

```
# Fill in the blank
___ PATH=$PATH:/home/ubuntu/kallisto_linux-v0.44.0
```

Listing 24 On AWS

```
wget ftp://ftp.ensembl.org/pub/release-67/fasta/homo_sapiens/cdna/Homo_sapiens.GRCh37.67.cdna.all.fa.gz
```

- Now we need to unzip the downloaded file:

Listing 25 On AWS

```
gunzip Homo_sapiens.GRCh37.67.cdna.all.fa.gz
```

- Now we can use Kallisto's index program to build the transcriptome index (N.B. this can take some time):

This command specified that we give this transcriptome index the name `hg19.ensembl`. *This will take a little while.*

Once complete we will have an *indexed human transcriptome* to match the reads against, we can then use Kallisto for fast transcript quantification on our FASTQ files.

Quantifying transcripts

To quantify transcripts with Kallisto we use the `quant` command option, specifying a transcript index (with the `-i` flag), an output directory (with the `-o` flag) in which to write results to file, and a list of our Fastq input files:

- Run transcript quantification for the pair of SRR2156848 FASTQ files:

This again will take some time (but not nearly as long as the alignment based mapping we did on our previous galaxy lab).

- Once run correctly, there should be three new files in the `SRR2156848_quant` directory:

Q. Now run the quantification calculation for the remaining samples:

Listing 26 On AWS

```
kallisto index -i hg19.ensembl Homo_sapiens.GRCh37.67.cdna.all.fa
```

Listing 27 On AWS

```
kallisto quant -i hg19.ensembl -o SRR2156848_quant SRR2156848_1.fastq SRR2156848_2.fastq
```

Listing 29 On AWS

```
# You complete these steps
kallisto quant -i ___ -o SRR2156849_quant SRR2156849_1.fastq SRR2156849_2.fastq

kallisto quant -i hg19.ensembl -o SRR2156850_quant ___.fastq ___.fastq

kallisto quant -i hg19.ensembl -o SRR2156851_quant SRR2156851_1.fastq SRR2156851_2.fastq
```

Scripting repetitive tasks

Rather than running `kallisto quant` separately for each of our remaining three samples, this is a good opportunity to write a simple **shell script**.

Scripting repetitive tasks is one of the most important habits to develop in bioinformatics. Not only does it save time and reduce the chance of typos creeping into individual commands, it also makes your analysis **reproducible** — anyone (including your future self) can re-run the entire analysis by executing a single file.

Just like writing functions in R, whenever you find yourself copy-pasting a command and changing only one small thing each time, that's a signal that a shell script with a loop would serve you better.

- Use a text editor (e.g. `nano`) to create a script called `run_kallisto.sh`:
- Add the following to the file, save, and exit (`Ctrl+O`, `Enter`, `Ctrl+X`):
- Make the script executable and run it:

The `echo` statements act as progress markers so you can track which sample is currently being processed.

Listing 28 On AWS

```
ls SRR2156848_quant
abundance.h5  abundance.tsv  run_info.json
```

Listing 30 On AWS

```
nano run_kallisto.sh
```

Check your results

You should now have a `*_quant/` directory for each sample.

Q. Have a look at the TSV format versions of these files to understand their structure. What do you notice about these files contents?

Transfer your results

With the transcript quantification completed, we are now finished our UNIX intensive work on AWS.

We can thus transfer our main results files to date (the various `abundance.tsv` files) back to our laptop for further analysis.

- On your computer (i.e. open a new Terminal) navigate to a directory where you want to store your results and use `scp` to transfer all your abundance results to this directory.

To make this command simpler to type you can use your **shell environment variables** from [earlier](#).

IMPORTANT: Stop your instance!

Please **Stop** your AWS instances at this point as we want to conserve funds for future work. To stop your instance:

- Return to the AWS console by visiting <https://awsed.ucsd.edu/>,
- Select your running instance,
- Click “Instance State” > “Stop” to suspend and stop being charged for resources.

Listing 31 run_kallisto.sh

```
#!/bin/bash
# Transcript quantification with Kallisto for samples SRR2156849-51

for id in SRR2156849 SRR2156850 SRR2156851; do
  echo "Processing sample: $id"
  kallisto quant -i hg19.ensembl -o ${id}_quant ${id}_1.fastq ${id}_2.fastq
done

echo "All samples complete!"
```

Listing 32 On AWS

```
chmod +x run_kallisto.sh
./run_kallisto.sh
```

Downstream analysis

Back on our laptop we can now use R and Bioconductor tools to further explore this large scale data-set.

For example there is an R function called `tximport()` in the **tximport** package, which enables straightforward import of Kallisto results.

With each sample having its own directory containing the Kallisto output, we can import the transcript count estimates into R using:

```
# BiocManager::install("tximport")

library(tximport)

# setup the folder and file-names to read
folders <- dir(pattern="SRR21568*")
samples <- sub("_quant", "", folders)
files <- file.path( folders, "abundance.h5" )
names(files) <- samples

txi.kallisto <- tximport(files, type = "kallisto", txOut = TRUE)
```

1 2 3 4

Listing 33 On YOUR computer

```
# Make sure you are accessing your keyfile and instance
scp -r -i ~/Downloads/bimm143w23.pem ubuntu@ec2-34-219-113-54.us-west-2.compute.amazonaws.com
```

Listing 34 On YOUR computer

```
# Change these to match your key and instance address:
export KEY=~/.Downloads/bioinf_barry.pem
export SERVER=ubuntu@ec2-34-210-54-74.us-west-2.compute.amazonaws.com

# Transfer results back to your current directory
scp -r -i $KEY $SERVER:~/*_quant .
```

```
head(txi.kallisto$counts)
```

	SRR2156848	SRR2156849	SRR2156850	SRR2156851
ENST00000539570	0	0	0.00000	0
ENST00000576455	0	0	2.62037	0
ENST00000510508	0	0	0.00000	0
ENST00000474471	0	1	1.00000	0
ENST00000381700	0	0	0.00000	0
ENST00000445946	0	0	0.00000	0

We now have our estimated transcript counts for each sample in R. We can see how many transcripts we have for each sample:

```
colSums(txi.kallisto$counts)
```

```
SRR2156848 SRR2156849 SRR2156850 SRR2156851
  2563611    2600800    2372309    2111474
```

And how many transcripts are detected in at least one sample:

```
sum(rowSums(txi.kallisto$counts)>0)
```

```
[1] 94561
```

Before subsequent analysis, we might want to filter out those annotated transcripts with no reads:

```
to.keep <- rowSums(txi.kallisto$counts) > 0
kset.nonzero <- txi.kallisto$counts[to.keep,]
```

And those with no change over the samples:

```
keep2 <- apply(kset.nonzero, 1, sd) > 0
x <- kset.nonzero[keep2,]
```

Principal Component Analysis

We can now apply any exploratory analysis technique to this counts matrix. As an example, we will perform a PCA of the transcriptomic profiles of these samples.

Now we compute the principal components, centering and scaling each transcript's measured levels so that each feature contributes equally to the PCA:

```
pca <- prcomp(t(x), scale=TRUE)
```

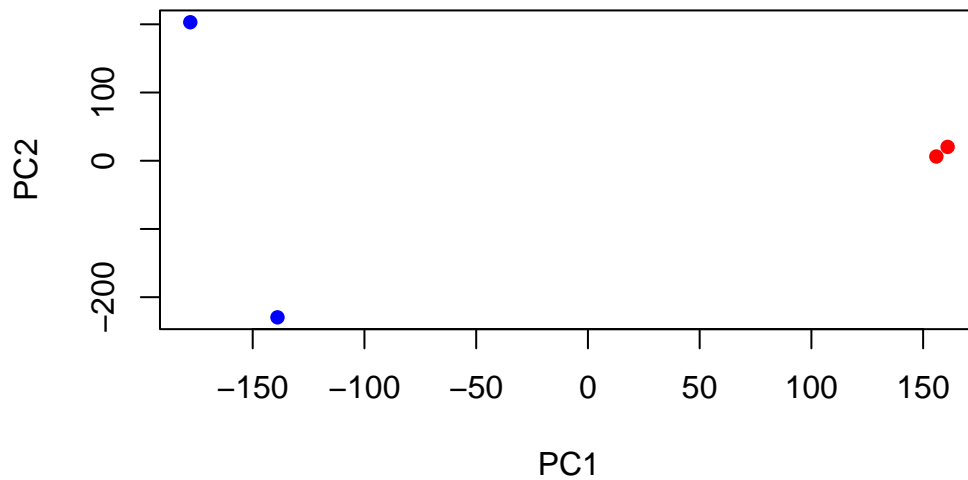
```
summary(pca)
```

Importance of components:

	PC1	PC2	PC3	PC4
Standard deviation	183.6379	177.3605	171.3020	1e+00
Proportion of Variance	0.3568	0.3328	0.3104	1e-05
Cumulative Proportion	0.3568	0.6895	1.0000	1e+00

Now we can use the first two principal components as a co-ordinate system for visualizing the summarized transcriptomic profiles of each sample:

```
plot(pca$x[,1], pca$x[,2],
     col=c("blue", "blue", "red", "red"),
     xlab="PC1", ylab="PC2", pch=16)
```

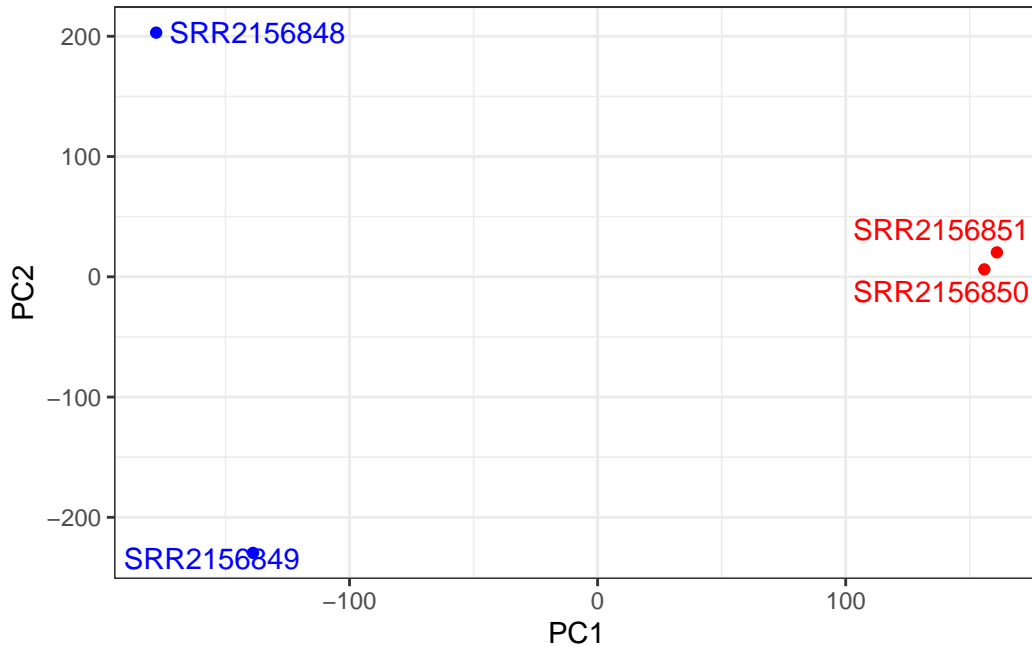


Q. Use ggplot to make a similar figure of PC1 vs PC2 and a separate figure PC1 vs PC3 and PC2 vs PC3.

```
library(ggplot2)
library(ggrepel)

mycols <- c("blue","blue","red","red")

ggplot(pca$x) +
  aes(PC1, PC2, label=rownames(pca$x)) +
  geom_point( col=mycols ) +
  geom_text_repel( col=mycols ) +
  theme_bw()
```



The plot makes it clear that PC1 separates the two control samples (SRR2156848 and SRR2156849) from the two enhancer-targeting CRISPR-Cas9 samples (SRR2156850 and SRR2156851). PC2 separates the two control samples from each other, and PC3 separates the two enhancer-targeting CRISPR samples from each other. This could be investigated further to see which genes result in these separation patterns. It is also at least slightly reassuring, implying that there are considerable differences between the treated and control samples.

OPTIONAL: Differential-expression analysis

We can use **DESeq2** to complete the differential-expression analysis that we are already familiar with:

An example of creating a DESeqDataSet for use with DESeq2:

```
library(DESeq2)
```

```
sampleTable <- data.frame(condition = factor(rep(c("control", "treatment"), each = 2)))
rownames(sampleTable) <- colnames(txi.kallisto$counts)
```

```
dds <- DESeqDataSetFromTximport(txi.kallisto,
                                sampleTable,
                                ~condition)
```

using counts and average transcript lengths from tximport

```
# dds is now ready for DESeq() see our previous classes on this
```

```
dds <- DESeq(dds)
```

```
res <- results(dds)  
head(res)
```

log2 fold change (MLE): condition treatment vs control

Wald test p-value: condition treatment vs control

DataFrame with 6 rows and 6 columns

	baseMean	log2FoldChange	lfcSE	stat	pvalue
	<numeric>	<numeric>	<numeric>	<numeric>	<numeric>
ENST00000539570	0.000000	NA	NA	NA	NA
ENST00000576455	0.761453	3.155061	4.86052	0.6491203	0.516261
ENST00000510508	0.000000	NA	NA	NA	NA
ENST00000474471	0.484938	0.181923	4.24871	0.0428185	0.965846
ENST00000381700	0.000000	NA	NA	NA	NA
ENST00000445946	0.000000	NA	NA	NA	NA
	padj				
	<numeric>				
ENST00000539570	NA				
ENST00000576455	NA				
ENST00000510508	NA				
ENST00000474471	NA				
ENST00000381700	NA				
ENST00000445946	NA				

These results could go on to be visualized and subjected to pathway analysis etc. as we have done in previous classes.