Differential Expression Analysis of Transcriptome sequencing data

Transcriptome sequencing (or RNA-seq) is an important tool in identifying gene expression differences between different conditions. Many researchers use RNAseq and differential expression analysis to identify the effect of a treatment on the expression of particular genes, or to identify how a species is interacting with an environmental variable (pathogens, water resources, nutrients etc).

In the major project for the Bioinformatics course, we are providing three datasets (1-3) from the same study in the model plant genome *Arabidopsis thaliana*

Genome-Wide Analysis of Gene Regulatory Networks of the FVE-HDA6-FLD Complex in Arabidopsis.

Yu CW, Chang KY & Wu K

FVE/MSI4 is a homolog of the mammalian RbAp48 protein. We found that FVE regulates flowering time by repressing FLC through decreasing histone H3K4 trimethylation and H3 acetylation. Furthermore, FVE interacts with the histone deacetylase HDA6 and the histone demethylase FLD, suggesting that these proteins may form a protein complex to regulate flowering time. **To further investigate the function of the FVE-FLD-HDA6 complex, we compared the gene expression profiles of fve, fld, and hda6 mutant plants by using RNA-seq analysis.**

http://www.ncbi.nlm.nih.gov/pubmed/27200029

Getting Started

Downloading the data & Setup

There are three separate datasets from this study, and each dataset file contains three parts.

1. Genome Data
    - *Arabidopsis thaliana* genome as a fasta file
    - *Arabidopsis thaliana* genome annotation as a gff3 file
2. Control Data
    - Three replicates of the wild-type/control strain of *Arabidopsis* called "Columbia" (Col)
3. Treatment Data
    - Three replicates of a hda5 gene mutant strain of *Arabidopsis*, either hda5-1, hda6-6 or hda9-1.

Due to some technical issues, we will rely on Rstudio to do a lot of the command-line work this week, as apposed to previously where you were using the ubuntu command-line for all your work.

Take the IP address from the VM list thats attached to this tutorial and copy-paste it into your web-browser with ":8787" at the end of the address.

For example:

`http://130.220.212.209:8787`

This should give you a Rstudio login screen in which you can login with your usual username and password (User: trainee, password: trainee). Once you are in, you will see the usual Rstudio setup as you're use to. For all of this work however, we are going to use the inbuilt shell command-line. To access the shell, go up to the "Tools" manu on the top of the page, and choose "Shell...". This will open a command-line type window.

Note: The Rstudio shell is ok, but it lacks a lot of the features that make unix and bash great. Like tab completion :(

However, it is handy to write all your scripts because Rstudio has an inbuilt script window which allows you the ability to write your shell scripts out without the terminal. I suggest using this

Creating your working directory

Because of the small amount of space on your VMs, we will be working on a mounted storage drive that is attached. This directory is in the base directory of your system:

```
# Change to the base directory
cd /

# Change into the directory /mnt
cd /mnt
```

The next two steps will need to be done by a user that has permissions to create directories on this area of the filesystem. Unfortunately the trainee user does not have adequate permissions, so therefore you need to login to the root user (ubuntu) and change permissions for you.

```
# Login to ubuntu (password = bioubuntu)
su ubuntu

# Create directory, change permissions, install two packages and exit the ubuntu user
sudo mkdir /mnt/trainee; sudo chown trainee:trainee /mnt/trainee; sudo pip install numpy; sudo pip install HTs
```

Now we have permission to use this area for our work! Change into the directory:

```
# Change into the new trainee directory
cd /mnt/trainee
```

Create a directory called tutorial where we will do our work.

```
mkdir tutorial

# Change into the tutorial directory
cd tutorial
```

First, we need to download the data, which might take some time (15-20 mins)

Now you can download the dataset assigned to you:

```
    wget -c "https://cloudstor.aarnet.edu.au/plus/index.php/s/ye78MxXCuaZwYlj/download" -O Dataset1.tar.gz

    wget -c "https://cloudstor.aarnet.edu.au/plus/index.php/s/mR3ZyDU3mFvfdvY/download" -O Dataset2.tar.gz

    wget -c "https://cloudstor.aarnet.edu.au/plus/index.php/s/kjv5kPhxgGITghk/download" -O Dataset3.tar.gz
```

```
This will now download the data into the current directory.
```

To extract the data in the dataset you will need to run:

```
    tar xvzf Dataset1.tar.gz
```

This will then give you all the files that you will need.

```
</br>
```

Scripting basics and making a pipeline

At the very basic, a bash shell script (like the ones that you have used and created in previous weeks) is just a list of commands that will run one after the other. This is the basic form of Bioinformatics processing script (which we refer to as a "pipeline").

Example1: Basic script

```
#!/bin/bash

command1 inputfile outputfile

command2 inputfile outputfile

command3 inputfile outputfile

..
```

For data processing tasks which use multiple input files, like tasks that you need to do for your project, you might need something a little more advanced or complex. You want to run the same tasks, but over multiple samples. For this, we can use a conditional loop, such as a "for" loop:

Example2: For loop

```
#!/bin/bash

# Iterate over each fastq.gz file in my directory
for file in \*.fastq.gz
 do

    command1 ${file} ${file}.output_command1

  command2 ${file} ${file}.output_command2

  command3 ${file} ${file}.output_command2

done
```

What this script does is loop over each sample in my directory that has the file extension "fastq.gz". It then reads each of those files one-by-one and executes the script as a block (like the one in Example1). After one file goes through the code block, it then goes to the next file in the directory and starts the code block again.

The loop in Example2 can also be written on one line by seaprating each line using a semi-colon (";"):

```
for file in \*.fastq.gz; do command1 ${file} ${file}.output_command1; \
command2 ${file} ${file}.output_command2; \
command3 ${file} ${file}.output_command2; done
```

For the purposes of this course however, we require you to space out your work like in Example2

In the end, you will have three output files for each sample in your directory.

In this tutorial (and assessment), we will get you to produce your very own pipeline, which will allow you to analyse all your dataset samples in one command!

His some additional resources that will explain for loops a little further:
http://ryanstutorials.net/bash-scripting-tutorial/bash-loops.php
http://williamslab.bscb.cornell.edu/?page_id=235

Starting analysis

Our pipeline will do four things, most of which you have done previously:

1. Run adapter and quality trimming
2. Align trimmed sequence reads to our reference genome; and lastly,
3. Count the number of reads that align to each gene region

Adapter Trimming

In this pipeline, we will trim adapters using a program called "AdapterRemoval". We will trim the adapters from our input fastq data (which is single-end data)

```
AdapterRemoval --file1 [file] \
               --output1 [file] \
               --gzip --trimqualities \
               --minquality 10 --minlength 20
```

The additional parameters that ive added will also trim poor quality bases from the remaining sequences:

```
--trimqualities       If set, trim bases at 5'/3' termini with quality scores <=
                      to --minquality value  [current: off]

--minquality PHRED    Inclusive minimum;
```

```
                      see --trimqualities for details [current: 2]

--minlength LENGTH    Reads shorter than this length are discarded following
                      trimming [current: 15].
```

We want to change these settings to create the best data possible. We dont want poor quality bases at the end of our reads, so we are adjusting the quality threshold to only allow bases > the "–minquality" parameter.

We also adjust the settings for the minimum length of the trimmed read. Short reads can sometimes lead to non-unique mapping, so we're increasing the "–minlength" parameter to 20bp

Alignments

For alignment we will use the program HISAT2, which is similar to the alignment tool that you've previously used (tophat), but much quicker and more accurate. Much like tophat, we need to first build the genome index before aligning our reads:

```
/opt/hisat2-2.0.4/hisat2-build Arabidopsis_thaliana.TAIR10.dna.toplevel.fa Athal_genome
```

Note: You HAVE to use the full path for hisat2 (one of the disadvantages of the Rstudio shell)

This command will produce your genome index with the prefix "Athal_genome". Each file starting with that prefix is used by the program to quickly search your input reads. Your alignment command is also fairly similar to tophat, however I have added an additional put at the end:

```
/opt/hisat2-2.0.4/hisat2 -p 2 -x Athal_genome -U [Trimmed fastq] | samtools view -bS -F4 - > [Output BAM]
```

HISAT2 actually outputs your alignment straight to screen (or what we call "Standard Out") in SAM format, which is good but takes up a lot of space on your VM. So what we can do instead is capture that SAM output and pipe that into the samtools program to produce a BAM file, which is compressed and much smaller in size. We also use the flag "-F4" to only output mapped reads, and therefore ignore the sequence reads that didn't map.

If you output the BAM/SAM file to a file, alignment statistics will be output to screen instead. This will give you information about how many reads were mapped etc. If you did not get this information, you can use the samtools command "flagstat" to generate this information:

```
samtools flagstat [Input BAM]
```

The VMs that you are using has 4 CPUs, meaning that you can actually make you alignments go faster by running your command with the "-p 3" command.

Quantification

Now we have a BAM file with all the information on which read mapped, and where they mapped. We can now use a program called HTseq-count to simply quantify the amount of reads that mapped to each gene region in the *Arabidopsis* genome.

To do this, we need to use the annotation information contained in the provided GFF3 file.

```
htseq-count -t gene -f bam [Input BAM] [Athal genome GFF3] > [Output Count file]
```

The file should look a lot like this:

```
AT1G01010       61
AT1G01020       41
AT1G01030       31
AT1G01040       81
AT1G01050       340
AT1G01060       15
AT1G01070       51
```

Each gene is in the first column, followed by the number of genes that mapped to that gene in the next.

And thats it for data processing! This data is now going to be used in further assessments for differential gene expression analysis in R

Assessment

Total: 20 marks

Assignment Task

1. Create a bash scripting pipeline that processes all your data in one command, with your raw data (fastq.gz files) as input and your gene counts as the output. The script must be able to be run in one command, and abide by the coding guidelines and principles that we outlined in Question 2 of Assessment 1. Additionally, we would also like to see statistics output as well:

   (A) How many reads are in each sample? (2 marks)

   (B) How many reads were trimmed in each sample using AdapterRemoval? (2 marks)

   (C) How many reads mapped? (2 marks)

   (D) What proportion of reads mapped to the genome in each sample? (2 marks)

   Valid, executible shell script (2 marks)

Long-form questions

2. Summarise the major platforms of next-generation sequencing machines and discuss their advantages and limitations. (5 marks)

3. Discuss the difference between a local (BLAST) and global aligners (Bowtie2/BWA). Additionally, what distinguishes a RNAseq aligner such as tophat or hisat2, to a aligner thats used for whole genome sequencing (such as Bowtie2 or BWA)? (5 marks)