RNAseq Analysis
GCAT-SEEK workshop

Mark Peterson

2015/May

Table of Contents

Chapter 1:
Computer setup

1.1. Background

RNA-seq analysis requires a lot of computer resources to handle the large datasets that are generated. Even a simple experiment can quickly run up near terabytes of data, and handling this data requires specially designed programs. Most of these programs can be run on your own computer, but they often take a very long time and slow your computer down to the point of no longer being usable. One simple solution to this problem is to run your analyses on a different computer, such as a Linux-based computer-cluster/supercomputer. In this module, we will walk through the process of using Linux for sequence analysis. In a later chapter, we will introduce a few alternatives, including web based platforms (See chapter 8).

One important thing to keep in mind: these tools are constantly changing as next-generation sequencing is still an emerging field. The instructions here will include details for the specific platforms and programs that will be used here, but will also explain what is happening, so that you can generalize to other applications. In general, remember that "Google is your friend" (GIYF) —someone else has almost always encountered problems similar to yours, and answers to their questions are usually available online with a little Google-fu. Second, most of these programs and computer resources come with extensive documentation that can walk you through many of the basic steps. When in doubt, "read the manual" (RTFM) and many of your questions may be addressed. All users, even advanced users, often do not remember the correct commands, but are just better at searching for the answer:

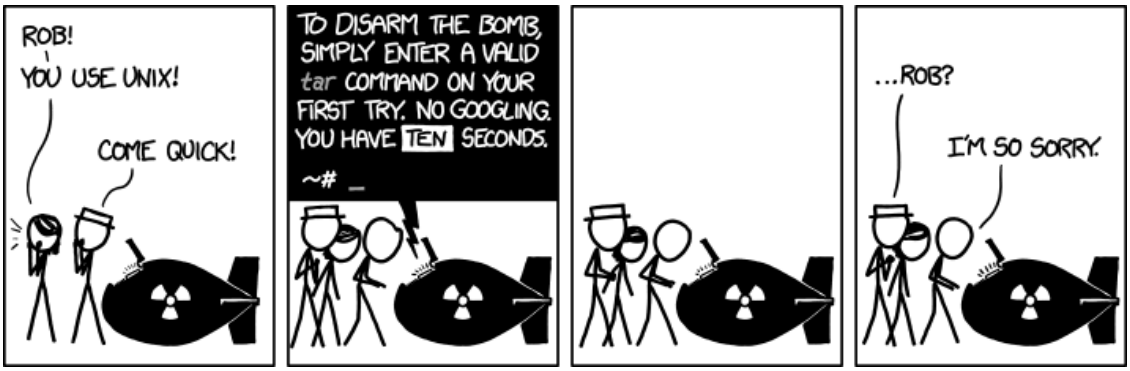

Figure 1.1: xkcd.com/1168 (Creative Commons Licencse)

1.2. Chapter goals

- Learn how to access a computer cluster or supercomputer
- Learn how to install necessary software
- Learn how to transfer data to and from these systems
- Understand a basic Linux command line interface
- Troubleshoot simple computer system problems

1.3. Sign on to a Linux computer system

Signing into a computer cluster or supercomputer is a bit more complicated than signing into your own computer. However, once mastered, it is straight forward to accomplish. You first need a system capable of "SSH" (Secure Shell) connecting. On Windows, you would need to install PuTTY to be able to gain access (search for "putty ssh" and follow the directions to install selecting "Windows installer for everything except PuTTYtel"). In Unix or Mac, the terminal comes pre-installed with this ability, just open a terminal window (`Ctrl+Alt+t` in Ubuntu, Applications → Utilities → "Terminal" in Mac). The basic command, which can be adapted to fill in the PuTTY screen is then:

```
ssh user@hostname
```

where "user" is your assigned user name on the system, and "hostname" is the address you are attempting to connect with: either an address or an IP address. Thus, for our system it will look like:

```
ssh user@mason.indiana.edu
```

You will then be asked for your password. Enter it here (the cursor will not move as you type), and you should be granted access to the system. The first time you log on to the super computer, it will likely ask you to select your shell. There is substantial debate about the "best" shell, but they will all function similarly for your needs. All of the examples in this workshop will use the BASH shell, so please select that one.

1.4. Linux tutorial

You will now have a command line prompt, and will be able to navigate as you would on any Linux system. We will go through detailed commands as we go through these modules, however, to get a feel for the Linux system, please work through this (wonderful) online tutorial: http://www.ee.surrey.ac.uk/Teaching/Unix/. A few notes on this tutorial:

- Some of the commands are specific to that University's file system, but they offer clear ways to work around that. For example, you may need to use the following to download one of the files:

  ```
  wget http://www.ee.surrey.ac.uk/Teaching/Unix/science.txt
  ```

- They are using a a different shell (tcsh instead of bash), which leads to a few small differences (e.g., the prompt in bash is '>' instead of '%' and in 8.4, the files they are referring to are called '.profile' and '.bashrc' instead of login and cshrc. Just a thing to be aware of.
- They suggest using 'nedit' to modify text files. But, it doesn't come standard on Linux. For this tutorial, replace 'nedit' with 'nano', which runs from the command line. Use `Ctrl+o` to save (for "out"), and `Ctrl+x` to exit.

1.5. Loading data

There are three basic ways we are going to get data on to computer for this class: scp, wget, and copying from things already loaded for you. First, let's try loading something from your local machine using `scp` (Secure Copy). Open a second terminal window on your computer

(Windows users, see the below for more detailed directions). You will then need to move (using "`cd folderName`") to a folder that contains the data you wish to upload, then enter the `scp` command. For example:

```
cd folderName
scp myFile.txt user@mason.indiana.edu:
```

In Windows, the command structure is a bit different. Access a terminal window by hitting `Ctrl+ESC`, typing "R", typing "cmd" and then hitting `Enter`. From here, `cd` to the appropriate folder, as above, then enter the command.

```
"C:\Program Files\PuTTY\pscp.exe" myFile.txt user@mason.indiana.edu:
```

In all places where you are told to use `scp`, you will have to use this full path instead. The path to the "pscp.exe" file must match the location of your installation (the folder you installed PuTTY into), so double-check the path if you get an error (and make sure you are using backslashes here, but only in Windows as they have not adopted the standard convention). You can also add that path to your "path" Environmental Variable to avoid having to retype it every time, but that is outside the scope of this module.

Note that quotation marks are necessary if there is a space in the directory name for any of the paths you are using (e.g. "Program Files" or "GCAT Workshop"), which is one (of many) reasons to avoid spaces in file and directory names.

Alternatively, you can use `wget` to download a file. The format is similar to `cp` and `scp`, just with a url in place of the source file and that the source file name is kept (but stored in your current directory). This command works with any download-able file, as you saw when working through the tutorial. We will use it most often for downloading programs to install, but it is also a nice way to download some data.

Finally, all of you have access to a shared directory, which we will be using extensively, named '/N/dc2/projects/GCAT/workshop'. Copy the file 'testFile.txt' into your own home directory, using `cd` and `cp` as necessary.

1.6. Where to go for help

The great thing about Linux is that there is a large community, so every problem you encounter has likely been faced (and solved) by other users who are happy to share their solutions. In addition, nearly every package comes with (at least) one of two simple ways to pull up more information:

```
man packageName
packageName --help
```

These two commands will pull up a lot of information about the syntax and options available for that software, often with usage examples. At the very least, it may identify the option you need help with: a great way to target your search for more information.

1.7. Further Reading

More information related to these topics can be found in:

- Any Unix/Linux guidebook, including great online resources
- The man and help pages of the package(s) you are interested in

Chapter 2:
Sequence processing and quality control

2.1. Background

After sending off your RNA for sequencing (or downloading it from SRA) and waiting patiently, you finally have data! . . . Now what? The files you have received contain billions of bases of sequence with information on quality, but where to begin? This module is the first step in that processing, and the first time we will be working directly at the command line interface with our data. Next generation sequencing platforms, by virtue of their large outputs, are bound to produce some errors. The goal of this module is to identify and remove many of these errors.

Figure 2.1: dilbert.com/strips/comic/2008-05-10/

2.2. Chapter goals

- Learn about sequencing quality scores
- Learn how to identify potential problems in sequencing data
- Learn how to handle and remove these potential errors
- Understand the indicators of quality sequence data
- Manage a large dataset without terror

2.3. Checking data quality

The first step of the process of data filtering is simply to see what you have. The data are provided to us in a format called "fastq" that includes a quality score, which provides information on how confident the sequencer is in its base calls. We will talk about this more in class, but the basic idea is that for Illumina runs, the score (Q) is calculated as:

$$Q = -10 * log_{10}P \tag{2.1}$$

Where 'P' is the probability that the base call is incorrect. Thus, a larger Q score indicates greater confidence in the base call. These scores are encoded along with the sequence information using the ASCII values corresponding to various characters with some offset to avoid the non-printing characters at the low end of ASCII values. As a general rule, scores over 20 suggest high quality data, while lower scores may be a cause for some concern. The steps below will walk us through how to identify those scores in a useful way.

This, like most of the linux commandline steps, will occur on the computer cluster, so connect, usign `ssh`, and let's begin First, lets just look at one of the sequence files. Move to the project directory, then use the command `less` to display the sequence file. Can you interpret those quality scores? Neither can I, so we will need to use some programs to help us visualize it.

2.4. Setting up fastqc

Running this from the command line the first time takes a few extra steps. Primarily because both java and fastqc will need to be installed. Both are slightly odd programs in that they do not require an "installation" but only require loading the program and adding a few variables to call them. First, we will install fastqc. All software that we download will be added to a directory named "opt" (for optional), which we must create.

```
mkdir ~/opt
```

Now, we will move into that directory and download the code for fastqc. We will then unpack (de-compress) it and delete the zip file. Finally, we need to set the permissions on the file to allow us to run the program.

Please note here that `rm` works rather differently than you may be used to. There is no "Trash" and there is no going back. Once you delete something with `rm` it is gone. So, it is generally good practice to use the "-i" flag, which stands for "interactive" and will ask you to confirm that you want to delete a file.

```
cd ~/opt
wget http://www.bioinformatics.babraham.ac.uk/projects/fastqc/fastqc_v0.10.1.zip
unzip fastqc_v0.10.1.zip
rm -i fastqc_v0.10.1.zip

cd FastQC
chmod u+x fastqc
```

For java, copy the file "jre-7u51-linux-i586.tar.gz" from the shared directory into 'opt'. You will then extract the file and remove the tar file. We will then unpack (de-compress) it and delete the zip file. Remember that tab autocompletion is a huge time and head-ache saver.

```
cd ~/opt
cp /N/dc2/projects/GCAT/workshop/jre-7u51-linux-i586.tar.gz .
tar -zxvf jre-7u51-linux-i586.tar.gz
rm jre-7u51-linux-i586.tar.gz
```

Now, we need to make it easier to run the program. Right now, it will only run if we type the full pathway, which is a pain. Instead, we are going to add the path to an "Environmental Variable" named PATH that tells the computer where to look for programs that you call. Importantly, it searches these directories in order, so if you have two identically named programs, it will only call the first one it finds. This also means we need to be careful not to accidentally remove part of the path. The easiest way to do this is to add the paths that we want to our variable, without deleting anything else.

```
export PATH=$PATH:$HOME/opt/FastQC:$HOME/opt/jre1.7.0_51/bin
echo 'export PATH=$PATH:$HOME/opt/FastQC:$HOME/opt/jre1.7.0_51/bin'  >>  ~/.bashrc
```

The first line adds fastqc and java to our path for this session, but that will be forgotten when we next log out. The second line adds that command to a file named ".bashrc" in your home directory, which is run every time you log on. Thus, from now on, FastQC and java will be in your PATH.

2.5. Running FastQC

Now, we are ready to analyze our data. Move into the 'seqData' directory in the project folder, this will keep all of our results together, instead of each running it in our own home folder. You will each be assigned to a sequence file(s), which will be yours to take all the way through this analysis. Each of you will execute the following, substituting your assigned file for `fileName` (note that throughout this document, purple text like this should be replaced with your information).

```
cd /N/dc2/projects/GCAT/workshop/seqData
fastqc -o fastQCresults fileName
```

For each assigned file. Once this has completed, it will place a folder named "fileName_fastqc" and a zipped version into the fastQCresults directory within seqData. From a terminal window, download the directory via `scp` by using the commands discussed in the "Computer Setup" module (See chapter 1) (Windows users, recall the workaround required from section 1.5). Note the period at the end of the line, which tells the computer "save the file right here with the same name. You can replace the "." with a different name for the file, or even a path to a different location, if you want.

```
scp -r user@mason.indiana.edu:\
/N/dc2/projects/GCAT/workshop/seqData/fastQCresults/file_fastqc  .
```

A few notes – you can leave the full scp line on one line, but it does not fit across the page here. If it is on one line, make sure that there are no spaces between the colon and the start of your path. Similarly, if you do leave it on two lines, make sure that there are no spaces between the colon and backslash (") or on the new line before the start of your path.

You can then open the results in a web browser by opening the directory and clicking on the file "fastqc_report.html". Look at the file, and see if there is anything you think needs to be done to clean up the data. Compare the results to others around you, and we will discuss them as a group.

2.6. Trimming bad data

There are two ways of filtering data: trimming ends that may have very low quality, or removing reads that are low quality. In general, short-read sequence aligners take quality information into account, and so conservative trimming and filtering is not necessary. However, if you have a run with very low quality ends, trimming those ends can help your analysis, especially if you are assembly a *de novo* transcriptome (see chapter 7 for more info). This program suite will likely be used later, and you will want it installed.

There are a number of tools designed to help you control read quality, each with their own benefits. For today, we will use a program called 'Trimmomatic' because it does a great job of explicitly handling paired-end data like these. This program requires java, which we already loaded, but follow those directions if you need to start from this point. Set up the program with the following:

```
cd ~/opt/
wget http://www.usadellab.org/cms/uploads/supplementary/Trimmomatic/Trimmomatic-0.32.zip
unzip Trimmomatic-0.32.zip
rm -i Trimmomatic-0.32.zip
```

To call this, we will use java, and simply pass the arguments we want to use. For more detail on each option, go to the website: http://www.usadellab.org/cms/?page=trimmomatic. One

note: paired-end data requires two outputs for each file, one for those that match the opposite direction read, and one for those that don't. The code below is an example that may be a helpful starting point; note that the '\' at the end of each line means 'put this all on one line; don't hit return yet' and can either be copied in directly (and interpreted by the console), or omitted to put everything on one line (interpreted by you).

Do not run this directly from the command line on the sequence files we have been using. The files we are using are large enough that you need to use a different system (a script submitted via `qsub`), which we will introduce in the next chapter (see section 3.6). If you would really like to use this, make sure to run the `head` command first to create a sample subset of the data (or, later, submit it as a script via `qsub`).

```
cd /N/dc2/projects/GCAT/workshop/seqData

head -n 4000 file.fq > smallExamples/subset_file.fq

cd smallExamples

java -jar ~/opt/Trimmomatic-0.32/trimmomatic-0.32.jar \
     SE -phred64 subset_file.fq\
     trim_file.fq \
     CROP:85 HEADCROP:4 \
     LEADING:3 TRAILING:3 \
     SLIDINGWINDOW:4:15 MINLEN:30
```

By line, these commands:

- Call the program
- The input data and information
  - 'SE' for single end data
  - the offset of the quality scores
  - the input sequence files
- The output trimmed sequence file
- Tell the program to keep the first 85 bases then cut the first 4 bases
- Cut the first (then last) bases, if the quality is below 3
- Check windows of 4 bases and cut the sequence when the average score goes below 15; only keep reads that are at least 30 bases long

We could then use these outputs in our downstream analyses.

2.7. Where to go for help

Most, if not all, of the programs that we use here have good webpages with substantial information on the usage of their programs (linked below). In addition, sites like "seqAnswers" and "BioStars" address many of the common issues that users encounter. Searches for these sites (or just general questions) will often lead you to individuals that have already addressed, and solved, the same challenges that you are facing.

2.8. Further Reading

More information related to these topics can be found in:

- http://www.usadellab.org/cms/?page=trimmomatic.
- http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

Chapter 3:
Read Mapping

3.1. Background

RNA-seq read mapping is the process of aligning reads against a genome (or transcriptome) in order to identify the position from which the RNA originated. This is another computationally intensive task, as it is generally attempting to align millions of reads against thousands of genes (or a full genome). In the case of alignment to a genome, the problem is further complicated by the fact that the sequenced RNA contains only exons, while in the genome these exons are substantially split by large introns. Even against transcriptomes, alternative splicing, skipped exons, and a number of processing steps can complicate the process.

Thus, many of the standard alignment tools that were developed for pairwise sequence alignment (e.g. Blast) are not suitable for this task. However, many alignment tools are emerging specifically designed to handle RNA alignment to genomes or transcriptomes. These tools use much shorter alignment matches and are designed to detect splicing and gene graphs. They work on similar principles to traditional alignment tools, and we will discuss them more as a group.

Figure 3.1: xkcd.com/676 (Creative Commons Licencse)

3.2. Chapter goals

- Learn about RNA alignment
- Learn the basic premise of different read mapping tools
- Understand the basic principles of RNA mapping
- Understand how to utilize various mapping tools

3.3. Choosing an aligner

There are several mapping tools available, and many are being developed specifically with short, and/or paired RNA reads in mind, such as bowtie and tophat from the tuxedo package. Bowtie is a fast read aligner, though it does not handle splicing explicitly. For aligning to a genome, Tophat uses initial bowtie reads to predict splice sites, and then aligns more reads across those junctions. Alternatively, if the transcripts are already known (through either transcriptome sequencing, like with our sample data, or gene prediction), bowtie can be used to align directly against known splice forms.

In this chapter, we will use a program called RSEM (RNA-Seq by Expectation-Maximization) that uses the bowtie alignment of reads to known transcripts. The program offers alternatives, including using the output from any aligner instead of using bowtie, that are discussed more thoroughly in the documentation linked below.

3.4. Installing RSEM

The RSEM package is a simple one to install, but requires several other programs (dependencies) to run completely. This installation will be broken into two parts, installing RSEM and installing those dependencies. First, move the directory in which you would like to install (~/opt), then follow these commands, which may be starting to look familiar now. The addition of the "`make`" command is necessary because RSEM needs to be compiled for the system's architecture. After running make, we add the path to our environment.

```
cd ~/opt
wget http://deweylab.biostat.wisc.edu/rsem/src/rsem-1.2.12.tar.gz
tar -zxvf rsem-1.2.12.tar.gz
rm -i rsem-1.2.12.tar.gz

cd rsem-1.2.12
make

export PATH=$PATH:$HOME/opt/rsem-1.2.12
echo 'export PATH=$PATH:$HOME/opt/rsem-1.2.12'  >>  ~/.bashrc
```

Next, we can add bowtie to our environment using a "module". This is a method that is slightly different than we used before, and its specific implementation will depend on the system you are working in. The basic idea is that updates to shared software (software available to all users) will be implemented more seamlessly this way. It is not available on all systems, but is a nice feature of some computer systems, as it will automatically load the program, and add it to our path, for us. As before, we are going to add the command to a profile to ensure that they are loaded every time instead of manually loading each time we sign in (or run a script).

```
module load bowtie
echo 'module load bowtie' >> ~/.modules
```

3.5. Prepare for alignment

We now have all of the software we need set to go, and can begin working on the process that will run the actual alignments. First, we need to generate our reference (either a transcriptome or genome) so that the program has something to align against. This should only take a couple

minutes, so we could run it directly from the command line. Move into the `seqData` directory. Please, do not run this in the workshop unless told to – having multiple iterations of this run simultaneously leads to corrupted files.

```
 Only one student should run this:
cd /N/dc2/projects/GCAT/workshop/seqData/transcriptome
rsem-prepare-reference --transcript-to-gene-map isotig_gene_map.txt \
                       --no-polyA  reference.fasta gcatRef
```

The files 'reference.fasta' and 'isotig_gene_map.txt', contain the transcriptome in FASTA format and the list of isoform to gene mappings. The 'gene_map' tells the program which transcripts belong to the same gene, which will allow us to analyze splice variants. The flag '--no-polyA' tells the program not to append poly-A tails to the transcripts. This is used because some of these transcripts may be partial, and should not include such tails. Finally, the last input 'gcatRef' tells the program where to write this file.

3.6. Run the alignment

The alignment we are going to run today will take several hours to complete (possibly days) when running on full size files. None of you likely wants to sit here with your computer connected for that long, despite how much you love this class. In addition, the some system will only allow jobs submitted from the command line to run for a set amount of time (in the case of Mason, 20 minutes) before it aborts them. This is done, in part, to make sure that large jobs are balanced on the machine. So, today we will introduce a new method for running commands on a computer cluster: a script and `qsub`. A script is just a series of commands to be executed (a simple computer program), and `qsub` is the system that is used to control the running of those jobs.

Remember that we are ~~lazy~~ efficient; so we are going to start by copying and modifying a sample script. Copy the file 'sampleScript.sh' from the project directory to your home directory. Run `less` to see what is in the file.

```
cp /N/dc2/projects/GCAT/workshop/sampleScript.sh ~/myFileName.sh
cd ~
less myFileName.sh
```

It should look something like this:

```
 Contents of file ``sampleScript.sh'':

  # @ job_type = serial
  # @ class = NORMAL
  # @ account_no = NONE
  #PBS -m e -l vmem=10gb,walltime=1:00:00
  # @ notification = always
  # @ output = batch.$(cluster).out
  # @ error = batch.$(Cluster).err
  # @ queue
```

The top three lines give information on how the job should be run and charged, and aren't anything we need to worry about. The line with 'PBS' give several commands to the qsub program, including to email you when the job is done (the -m e), how much RAM to let you use, and

how long to let the job run. The `vmem` and `walltime` commands are required, as this is what the system uses to balance the jobs that are submitted. The rest of the information tells the system how to save your outputs, and that the job should go to the queue.

Below this, we are going to insert our commands. These can be anything that you would submit in the terminal, and the same syntax and rules apply, including spelling and capitalization, so be careful. To add these commands, we are going to use a word processing program: 'nano'. Run the command '`nano`' followed by the name of your script. It will open the document for editing in a simple editor.

```
nano ~/myFileName.sh
```

The instructions at the bottom of the screen tell you how to run the program, but it is similar to notepad, just without the click-able menus. To save a file, type '`Ctrl+o`', name the file (or accept the current name) and hit enter. To close the program, type '`Ctrl+x`'. Move to the bottom of the file (the mouse won't work) and add the following commands:

```
 Add this to your script using: nano ~/myFileName.sh

cd /N/dc2/projects/GCAT/workshop/seqData

rsem-calculate-expression --phred64-quals file.fq \
                          transcriptome/gcatRef align/file
```

The second line (wrapped above) calls the aligner, and runs the expression calculation. First, we tell it that the file has an offset of 64. Next, we give it the name of our file to analyze; do not forget to change the purple "file" portion to match your file name. We then tell it which reference to use (the one we created above), and finally the output name, which includes the directory where we want to save it, and the name of your assigned sample. The directory 'align' should already be created to ensure that your outputs are saved in the correct place (create it using `mkdir` if it doesn't exist). Save the file (`Ctrl+o`) and exit (`Ctrl+x`).

You will notice that this line only works on one sample at a time. If you want to run this for multiple files, you will need to write a line like the above for each of your samples. Once you are more comfortable with the Linux syntax, you may also want to explore GNU Parallel. However, that syntax is beyond the scope of this tutorial.

Now, we will submit the job with this command:

```
qsub ~/myFileName.sh
```

and now we wait. We can see the progress (if any), by calling '`qstat`' which shows all running and queued jobs. If your run:

```
qstat -u userName
```

You will see only your submitted jobs. Run `ls` in your home directory to see if there are any output files yet, and use `less` to view them. They should be similar to the outputs you would expect to see in the terminal.

3.7. Where to go for help

Many computer systems use a very similar job manager, and so a search for most of the keywords we have used in this chapter may help solve the problem that you are facing. In addition,

the man pages for '`qsub`' and '`qstat`' contain a great deal of information on the submission of jobs. Finally, the 'Further Reading' includes the websites for each of the programs we used today, and they have extended discussions of many available options.

3.8. Further Reading

More information related to these topics can be found in:

- http://tophat.cbcb.umd.edu/manual.shtml
- http://bowtie-bio.sourceforge.net/index.shtml
- http://bowtie-bio.sourceforge.net/manual.shtml
- http://cufflinks.cbcb.umd.edu/
- http://deweylab.biostat.wisc.edu/rsem/
- http://deweylab.biostat.wisc.edu/rsem/README.html

Chapter 4:
Differential Expression

4.1. Background

Now that the reads have been quality checked, assembled, and aligned, there is finally a bit of biology that can be directly addressed. In this module, we will start by examining differences in gene expression between our sample groups. The analysis today will use just one of the programs currently available to assess gene expression differences (and more are constantly emerging). We will discuss a few of the fundamental principle of gene expression and use this program as an example. However, you may find that you like the interface (or assumptions) of a different program better. There are plenty of options available.

The commonality to all of these approaches is that they all attempt some form of normalizing for read count (we accomplished this once already by calculating fpkm along with raw read counts) and then use a combination of modeling and basic statistics to determine if two groups differ in expression. This raises a very large problem of multiple testing: with a p-value of 0.05, we expect 5 in every 100 tests to be false positives. Obviously, when dealing with thousands of genes, that number overwhelms the real results. There are a number of proposed corrections (choosing the top X most-significant genes a priori, setting a stricter p-value a priori, etc.), but we are going to focus today on the False Discovery Rate (FDR). FDR calculates an acceptable rate of false/ to true/positives in a data set, rather than relying on raw p-values. It uses the p-values strike this balance, usually at 0.05, or 1 in 20 positives expected to be false-positives.

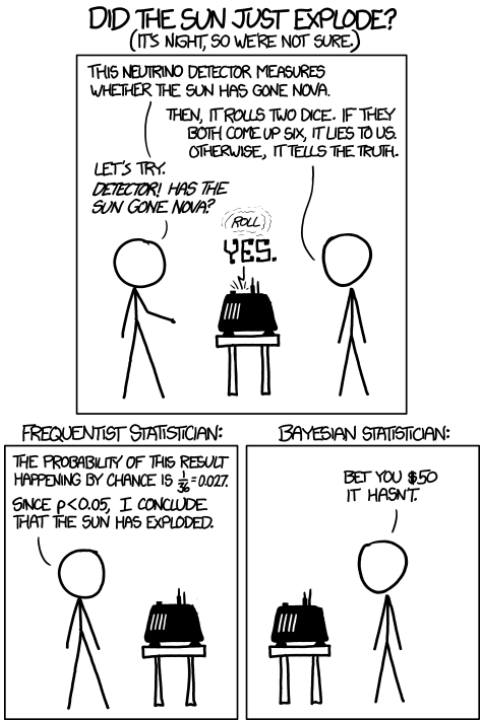

Figure 4.1: xkcd.com/1132 (Creative Commons Licencse)

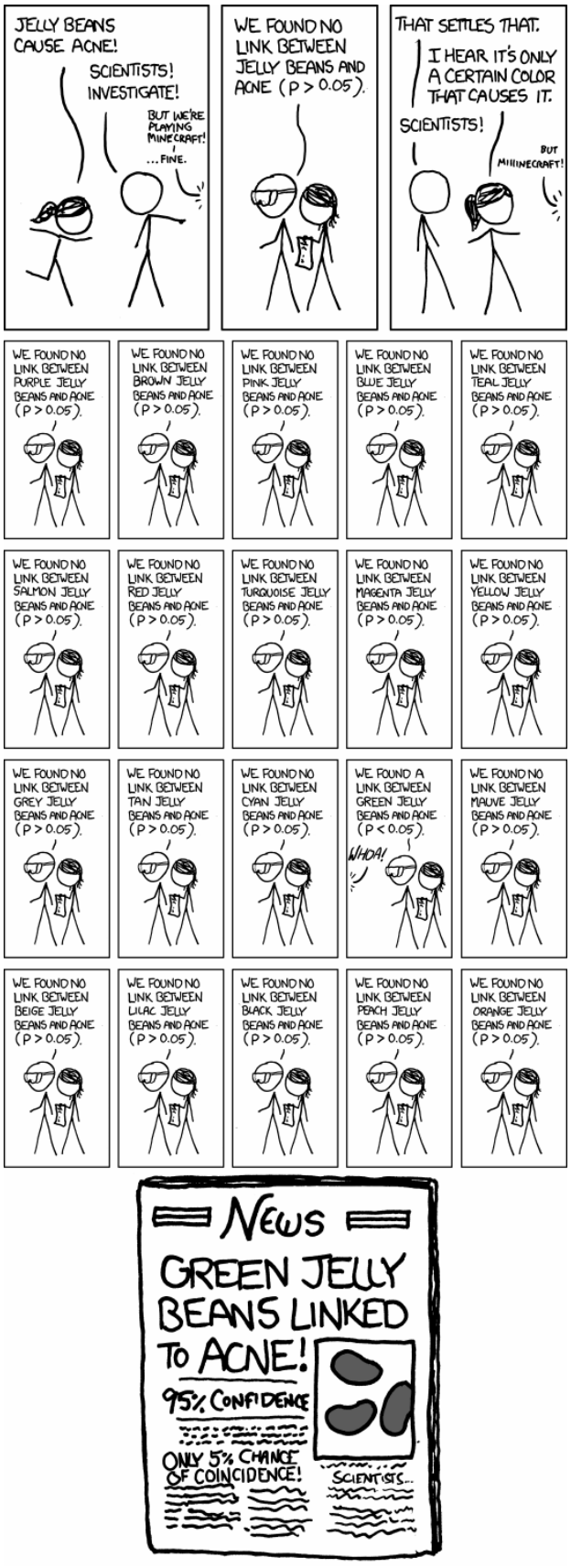

Figure 4.2: xkcd.com/882 (Creative Commons Licencse)

4.2. Chapter goals

- Discuss the concepts behind gene expression analysis
- Discuss statistical balance between false/positives and false/rejection
- Learn the basics of (at least) one differential expression analysis program
- Understand the basic principles managing large datasets
- Understand the statistical principals of large datasets

4.3. Prepare R

This chapter will utilize the R statistical environment to conduct our analyses. R is an incredibly powerful statistics package designed with high quality graphics explicitly in mind, which makes the default plots it produces elegant while allowing users full control over the outputs. In addition, this flexibility extends to the handling and display of raw data, making it possible to store, manipulate, and analyze a wide variety of data types in a single program. This flexibility comes at the cost of a large learning curve, especially as many of you are probably new to computer programming. Finally, R is free, both free as in beer (no cost) and free as in speech (the source code is available, can be manipulated, and can run on any platform), making it an accessible choice for students and researchers.

If you have not used R before, please consult an R tutorial before proceeding, such as the one provided with this document. Without it, you will still likely be able to complete this section, but you will struggle to understand the syntax that is being presented. At the least, you will need to install R, available, with instructions, at: http://cran.r-project.org/bin In addition, it is also strongly recommended that you install the graphical interface RStudio to make R more comfortable to use, available at: http://www.rstudio.com/ide/download/desktop.

4.4. Download data

A note to users looking to run this type of analyis on a complete dataset. You will have noticed that, to this point, we have been analyzing each of the sequence read files separately. This is great for learning what is happening, but is not so great when you are trying to analyze a full experiment, which may have anywhere from 6 to 100 samples easily. There are, however, much faster ways to analyze multiple samples, in parallel, though they are outside the scope of this manual. For an example of such scripts, using these data, please see the git repository `https://bitbucket.org/petersmp/publishedtutorials/`.

Many of the tools for differential expression can be run directly from the command line, including the steps that we are going to run below. However, it is often easier, especially when first running analyses, to run them on your local machine. Here, we will download the count data, and start running some basic analyses on differential expression. To do this, first move all of the results of interest (here, the "genes.results" files) into a single directory, to make the download easier. Because we will be analyzing all of the samples, this only needs to be run once, not by each participant.

```
 Only one student should run this:
cd /N/dc2/projects/GCAT/workshop/seqData
mkdir geneResult
cp align/*genes* geneResult/
```

This copies all of the files containing the word "genes" from the "align" directory into our newly created geneResult" directory'. Now, on your local machine, move (cd) to a directory for this workshop, then download that entire directory, and the annotation information with:

```
 On your local machine:
cd /path/to/your/directory/
scp -r user@mason.indiana.edu:/N/dc2/projects/GCAT/workshop/seqData/geneResult .
scp -r user@mason.indiana.edu:/N/dc2/projects/GCAT/workshop/seqData/annotations .
```

Which will download our "geneResult" and "annotations" directories to where you are (note the "." which again tells scp "right here").

4.5. DESeq

Several alternatives exist, including other command line options, such as RSEM, cuffdiff, and a number of packages in R, including limma, cummeRbund, EBSeq, and DESeq. In this section, we will focus on the R Bioconductor package DESeq, which is fast, flexible, and which I have used extensively. DESeq utilizes a negative binomial distribution and offers several alternative approaches for significance testing. Covering the theoretical background of this package, especially in contrast to others, is beyond the scope of this workshop. Instead, please refer to the publication describing DESeq and comparing it to other methods (see section 4.8).

The general steps of the DESeq pipeline are:

- Combine count data (such as from RSEM) into a single file
- Normalize the read counts to account for sequencing depth differences
- Estimate variance (called dispersion) for each gene
- Compare expression between groups, and
- Calculate significance

Each of these steps are relatively common, though there is some discussion over the best way to handle dispersion and significance. For a more detailed description of these steps, and the underlying theory, please refer to the vignette for DESeq (see section 4.8). For this workshop, along with other projects, I have written an R package, `rnaseqWrapper`, that wraps these steps into a single function, with sensible defaults. For those of you accessing these materials from outside of the workshop, please contact me directly for the most updated version of the package.

4.5.1. Install rnaseqWrapper

To install this R package, open RStudio (this will also work in R directly) and follow these steps (only necessary the first time you install it, or if you update it). Note that the additional dependencies are only required by a few of the functions, but that includes the ones you will be using in these chapters. This will likely take five to ten minutes.

```
## Install MPP's rnaseqWrapper package
install.packages('rnaseqWrapper')

## Install DESeq and topGO from Bioconductor
source("http://bioconductor.org/biocLite.R")
```

```
biocLite(c("topGO","Rgraphviz","DESeq"))

## To check if installed correctly
## This is the only line you will need to run to load the package in the future
## When this loads correctly, a message will list what is loading
library(rnaseqWrapper)
```

4.5.2. Running DESeq with rnaseqWrapper

As before, we need to start by loading in our expression data, using the outputs from RSEM. So, use `setwd()` to navigate to the directory in which you saved the data files above. You will likely want your working directory to be (at least) one level above the directory where you saved the data, largely to make it simpler to create analysis output directories, but that is just personal preference of mine. Once the data are downloaded, the following function (part of my rnaseqWrapper package) will automatically read in and merge the files together, naming each column using the file names. Read the help file for more information if you are interested (`?mergeCountFiles`).

```
## Load the data you just downloaded to your computer
#  Note, you will likely have saved the outputs in a different directory
setwd("path/to/data/")
countData <- mergeCountFiles("geneResult/")

# View the data
head(countData)
```

As you can see, the files are now merged, and we can work with them together. Now, DESeq expects count data, as its internal normalization, and the method it uses to determine significance, rely on raw counts. The SEM column "expected_count" is roughly raw counts, with just a small amount of uncertainty given for the assignment of reads. As such, we can use those columns for DESeq, but we need to let DESeq know that they are counts by rounding the data to integers. To extract those columns, we will use the function `grep()` which searches for matches between its first argument (e.g. "expected_count" below) in its second argument (e.g. the names of the columns) and returns the index (position) of the matches. I also like to remove the common portion of the column names, using `gsub()`, which adds an argument to replace the matched portion of the text. Both accept regular expressions, which makes them very powerful, but also a lot more confusing. For now, just know that `gsub()` is a nice way to systematically change things, and that you can replace the empty quotes ("") with different text if you just want to change the names (e.g. to "_reads").

```
## Extract just the read counts, and round them
myCountData <- round( countData[,grep(".expected_count",names(countData))],0)

## Trim the names to make the plots a bit nicer:
names(myCountData) <- gsub(".expected_count","",names(myCountData))

head(myCountData)
```

Finally, we can run DESeq on these rounded data. For now, the basic defaults will work just fine, but read the help documentation for more information on how you might tune this for your own project. For now, it is enough to know that the function takes your count data, searches for

the "conditions" (e.g. sexes or treatments) in the column names, runs a differential expression test, and saves the results (both as an R object, and in your working directory, preceded by the "outNamePrefix"). If your samples don't include your treatment names, you can also specify "conds", which lists each sample's treatment.

```
## Run DESeq
deOut <- DESeqWrapper(myCountData, # Our count data to use
                      conditions=c("male","female"), # conditions to compare
                      outNamePrefix="DESeqTest/") # Where to save the outputs
```

This function goes through all of the basic steps of DESeq. Occasionally, especially with very small or low-count data, this function will throw an error stating that the dispersion method failed. This means that the default included did not work, and may indicate problems with the underlying data. In the future, the package will offer alternatives for this step, and other steps, but for now, you will have to work through each step separately, in order to solve this problem.

The function saves the outputs in a few different, useful, formats:

- A series of pdf plots that show some basic characteristics of the data
- Tab-delimited output of the differential expression test
- Tab-delimited output of the normalized read counts (if requested), and
- An R object which we can directly manipulate.

The first few items are all useful for record keeping and visual inspection. Open up the directory where you saved the data, and look at the plots it generated. The last item holds all of the generated data (but not plots) in a single object, which allows us to explore our results in more depth. So, we will save it now to make it easier to reload in the future (instead of having to use `read.table()` for each element) when you start analyzing our data. Use the below code to save the R object.

```
## Save the DESeq R object:
save(deOut,file="DESeqTest/deOut.Rdata")
```

Before exiting R, spend a few minutes playing with the data you have just generated. Make some plots, see what is available, and just generally fiddle. As you read the next section, and we discuss it, play with the data you have, and think about what you might do with it to meet the objectives of your study.

One place to consider starting is by generating a heat map, one of the standards in gene expression analysis. R makes this very easy to do, and the `rnaseqWrapper` package includes a wrapper to make it (somewhat easier). A similar plot is automatically generated by `DESeqWrapper()` and saved in the pdf output, but this will give you the power to tweak the plot to your purposes.

```
## Save oval for easier use
myQvals <- deOut$deOutputs$malevsfemale$padj

## Limit data to interesting genes
toPlot <- as.matrix(myCountData[myQvals < 0.05 & !is.na(myQvals),])

## Make a heatmap
heatmap.mark(toPlot)
```

```
## Add color and a legend to show more of the options
heatmap.mark(toPlot,
             cexCol=.9, # make column labels smaller
             ColSideColors = rep(c("red","blue"),each=8), # match column order
             scaleLabel="") # turn off label to leave room

legend(x="topleft",inset=c(-.01,.13), # where the label should go
       bty="n", cex=.5, # no box around it, and set the size
       legend=c("female","male"), # What should be there
       fill=c("red","blue"), # Colors to use
       title="Conditions") #Label for the legend
```

4.6. Interpreting differential expression analysis

This section provides a thin background on differential expression analysis that will be used to guide our in class discussion. Hopefully, the details in this section will help to guide your decision making for your own data interpretation.

There are a plethora of gene expression analysis tools, and there seem to be more emerging every week. Some of these are just small tweaks to increase speed or user/friendliness, but some make major changes to the underlying assumptions and math. In particular, different differential expression tools rely on different mathematical models (e.g. normal vs. negative binomial distributions) to estimate the probability that identified differences are real. The choice of which program is best for you and your data is unique, and must address ease of use, mathematical assumptions, the details of your project, and what you are attempting to achieve. For most analyses, the results will be relatively similar, and there are few bad choices.

All of these programs, and likely all of the ones still to come, will give outputs in a similar fashion. A table with each analyzed gene as a row. The columns will represent the basic statistics and may include: average expression level (for each group or for the two combined), difference between the two (fold-difference, or log-fold-difference), a statistical test, and the FDR level of significance. No matter what your specific project, focusing on the significant genes (those below an a priori FDR cutoff) and the direction of their difference is likely to be most fruitful. Many programs will write just these genes to a separate file to make this analysis easier.

From here, there are many options to progress. One is Gene Ontology (GO) analysis. If your species annotation includes GO terms, you can determine which GO terms are over/represented. As above, there are many programs for doing just this and they all work on similar principles. The idea is to find GO terms that are more common among significant genes than among the full set of annotated genes. That is, a GO term that is present 5 times among significant genes (that is, 5 of the significant genes are annotated with that GO term) is more likely to be meaningful if there are only 5 genes with that annotation in the full set, than if there are 50 genes with that annotation. GO analysis provides a nice snapshot of what is being regulated differently between your two groups.

Alternatively, you can focus on individual genes that are differentially expressed. This is particularly fruitful if you have an a priori reason to expect a gene to vary. For example, if your sample groups are a model for a disease or developmental state, you may expect known marker-genes to differ between the groups. Identifying such differences can help to confirm the validity of your experimental set up. In addition, you can search through the gene lists for genes that may play interesting and related functions. Identifying these genes may help to guide future research projects or to connect your findings to those of other research systems.

These approaches are neither exhaustive, nor mutually exclusive. The simplest answer to "what should I do now?" is to play with your data and see what emerges. You may be surprised

to find a set of key genes from another system (another disease, for example) differentially expressed between your groups. Breaking down the gene lists (e.g., by GO term, by pathway, or even randomly) will allow students a great deal of autonomy and may lead to some surprising and insightful conclusions as they dig deeply into a subset of your analysis.

4.7. Where to go for help

Differential expression analysis, despite being studied for a long time now (qPCR, microarrays, Northern blots, etc.), is still an unsettled field. Especially as datasets grow (now analyzing tens of thousands of genes simultaneously), there is great debate over many basic questions. How can we best control for known and unknown sources of error? How should multiple-testing be corrected? How can we ensure we don't reject too many true differences? Does fold-change matter?

The downside is that there are many open questions and there are rarely simple answers. You may need to analyze your data in several different fashions in order to figure out what works best for your system, questions, and needs. The upside is that there is not one "right" way to do things, which allows both you and your students substantial leeway to explore your data in novel ways. Who knows, your approach may even revolutionize the field.

All this is to say: the answers for how to best analyze your data probably lie in your data, not with someone else. Answers to simple questions abound on places like SeqAnswers, and in the documentation to your programs of choice. However, you may need to try several different approaches before the answers you get seem to completely address the questions you started with. List-serves and outside guidance are great to make sure that we are not fooling ourselves, and should be utilized whenever possible. Just, don't expect a clean answer.

As a final note, it is easy with large datasets to fool yourself. It is important to get critical external feedback to your methods. Simulating your data analysis can also be a great way to make sure that your results are not just a product of some unseen bias. I came three days away from submitting (in a revision no less) an analysis that was, quite frankly, completely wrong. I had simulated at a downstream step, and missed an underlying artifact in my data. The conclusion, around which I had based most of that paper and several talks, was just flat wrong. It was an artifact and did not represent anything meaningful. Be careful in your analyses, and always listen when others raise concerns about your approach.

4.8. Further Reading

More information on the `DESeq` package, including the full package vignette and a link to the full paper (available open access at the link below) can be found at:

- http://www.bioconductor.org/packages/release/bioc/html/DESeq.html
- http://genomebiology.com/2010/11/10/R106/

Chapter 5:
Functional group analysis

5.1. Background

After all the hard work of generating RNA-Seq data, one is rewarded with a (hopefully) long list of genes differentially expressed between treatments, or mutants, or tissues, or whatever you were testing for. . . . Now what?

Human nature expects highly meaningful genes to leap to the top of these hard-earned lists, but in reality there are usually hundreds to thousands of differentially expressed genes and too much information to easily comprehend or summarize. How can we extract meaningful information from such large lists?

One approach is to consider what kind of genes are being affected. From a gene list, especially a long one, it is difficult to see a larger pattern of gene types being affected, in part because it would require knowing what every gene does. This is where Gene Ontology (GO) annotation comes in. The GO system is a heirarchical set of gene function annotations, which has become the de facto standard across all species. This works because the functions are defined in a way that applies from plants to animals to bacteria.

This annotation is assembled into a "graph" to show the ways that functions are related. For example, a hormone receptor might be annotated with the function "response to hormone" and when more is learned, further assigned to the more specific function "response to steroid hormone." Further, genes can be assigned to any number of functions, reflecting the wide range of roles that any given gene might play. Our hormone receptor, for example, might also be annotated as "sequence-specific DNA binding" if it plays a role in regulating gene expression.

By utilizing this type of annotation, we can move beyond a list of (often cryptic) gene names. A list of affected functions is much smaller, and often much more intuitive to interpret. This can make presenting a coherent story around your data much simpler, and is a great guide for both the investigator and readers. In this chapter, we will cover the basics of these analyses, and how to conduct them.

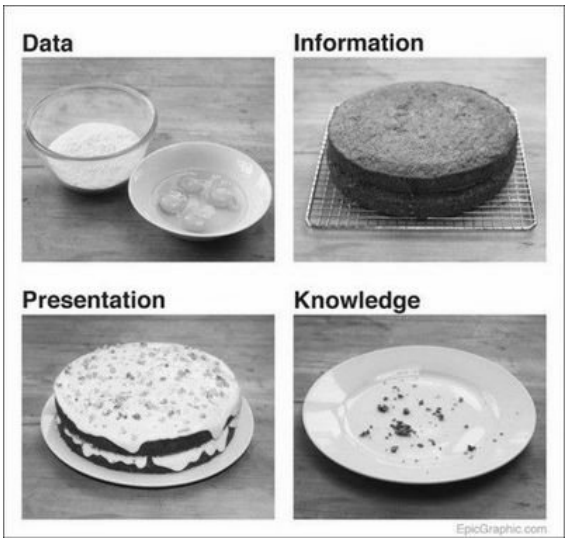

Figure 5.1: epicgraphic.com/data-cake/

5.2. Chapter goals

- Understand the basics of GO annotation
- Understand the statistics of GO analysis
- Be able to run a GO analysis
- Be able to interpret the output of a GO analysis

5.3. Basic statistics

Functional group analysis aims to determine which GO annotations are enriched (over-represented) in subsamples of gene lists (e.g., differentially expressed genes; referred to as "DE" below). Functional annotations that appear disproportionately among DE genes are likely to play a role in the phenotypic response to your test; those that appear with equal probability in DE genes and all genes are not.

In Table 5.1, we see that we are trying to show that there are more genes in the "A" position, than expected by chance. Put simply: we want to know if the ratio $\frac{A}{B}$ is significantly greater than the ratio $\frac{C}{D}$. In this way, a GO function assigned to 11 genes in the dataset, 10 of which are significantly DE, is much more meaningful (and over-represented), than a function assigned to 300 genes, 10 of which are significantly DE.

	is Sig DE	not Sig DE
Has GO term	**A**	B
Not this GO	C	D

Table 5.1: Depiction of functional group testing

To put this more formally, functional analysis asks whether the distribution of GO annotation and significance are indepenent. These analyses generally use familiar statistical principles for contingency tables, such as Chi-square and Fisher's exact tests. The p-values from these tests reflect the probability of enrichment of particular functional groups among DE genes. However, because many tests are being performed, we must adjust for multiple testing, generally using false discovery rate correction (similar to what we used in Chapter 4).

One important point to consider is what set of genes should be included in the background (full) set. It is tempting (and often a default for many programs) to use all of the genes in the genome as reference set. However, experiment-specific detected genes are almost always more appropriate background lists. Because one tissue may not express all genes, the use of a genome-wide background will then also reflect tissue specificity, which is presumably of less interest. In addition, if many genes from a functional group are not expressed, they cannot be DE, which means that, even all of the expressed genes in the group are DE, the GO category may still not be identified as over-represented.

Finally, be aware that the statistical power of functional group analysis is highly dependent on the number of genes in the selected list. Lists of 1 to 10 genes have little power to detect enrichment of functional groups whereas 300 to 3,000 DE genes are ideal in experiments with a background list of 5,000 to 20,000 genes.

5.4. Getting GO annotations

Unfortunately, despite the standardization of the GO system, the identification of GO terms for your organism can still be tricky. Each study system is different, each has their own repositories, and those may (or may not) be directly linked to the gene names in your particular reference. So, head to trusty google, and start looking for resources for your organism. If you hit problems, seek out assistance from colleagues, or list serves (including the GCAT-SEEK listserve). Others have probably faced (and overcome) the same hurdles you face, and they are often happy to help.

This problem is worse in species with limited genomic resources, as non-model species have little or no GO functional annotation. This lack can be overcome by determining the best hit (generally at the protein level using blastp) for each gene against a (hopefully closely) related model species (such as Drosophila, Arabidopsis, Mus, etc.). Tools, such as Blast2GO, can greatly simplify this process, but know that it is likely to take a long time.

Results carry the assumption that blastp homology confers meaningful information about function across distantly related species. Conservation of gene function is sufficiently strong that this is a fairly safe assumption. However, inferences about the function of specific genes remains putative and should be treated as such. The key is to recognize both the possibilities and the limitations, and neither discard the bad because it isn't perfect, nor over-interpret the good as if it were perfect.

5.5. Run a GO analysis

Finally, we are ready to run a GO analysis. There are a lot of tools available to run these analyses, including graphical programs (such as BiNGO in Cytoscape) and web based tools (such as David, described in Section 8.6). A full, and very long, list of such tools is available at: http://neurolex.org/wiki/Category:Resource:Gene_Ontology_Tools, if you are interested. However, today, we will be working with the R package `topGO` from BioConductor.

In RStudio on your computer, load the `topGO` and `rnaseqWrapper` packages, which you installed in Section 4.5.1 (remember that a package only needs to be installed once on each computer).

```
## Run GO analysis
## Mark Peterson 2014 June 06

## Set working directory
setwd("path/to/data/")

## Load topGO package and rnaseqWrapper
library(topGO)
library(rnaseqWrapper)
```

Then we will load our annotations, which you downloaded with the RSEM results. This includes a list of GO annotations developed for the dark-eyed junco, in a format usable by `topGO`. Your GO annotations may come in a different format, so take note of the structure in the tsv file to guide you.

```
## Load the GO annotation file
goAnno <- readMappings("annotations/goAnnotation.tsv")
```

We also need to load in the DESeq results (saved in Subsection 4.5.2).

```
## Read in DESeq output
## This loads in an R data object, and returns the name of the object(s)
## So, "deOutName" has the name of the data we actually want to use
## Here, that is our DESeq object "deOut"
deOutName <- load("DESeqTest/deOut.Rdata")

## Set an easier name to access the part you are interested in:
myDE <- deOut$deOutputs$malevsfemale
```

Now, we need to identify our "expressed" background.

```
## Which Genes were 'expressed' (using mean expression over 5 counts)
expGenes <- myDE$id[ myDE$baseMean > 5 ]
```

. . . and our significantly DE genes. Note the use of "`!is.na(myDE$pval)`" This tells R to ignore unassigned p-values (such as for genes with no counts), otherwise R doesn't know what to do with NA values, and returns a lot of "NA" to warn us of that.

```
## Which genes are significantly DE?
## Normally, you would use the adjusted pvalue (but we have low power)
sigGenes <- myDE$id[ myDE$pval < 0.05 & !is.na(myDE$pval) ]
```

Now, we can run a basic GO analysis, using a wrapper from the `rnaseqWrapper` package. The function `runGOAnalysis()` takes our lists of genes, and GO annotations, and runs all of the tests for us. It defaults to running the "Biological Processes" ontology, but can also be set to analyze "Molecular Function" or "Cellular Component." Check out the help for more details on available options. Here, we are going to run the analysis with the "classic" algorithm, which works just as we described above: running a Fisher's exact test on every GO term we are analyzing.

```
## Run GO analysis
goOutput <- runGOAnalysis(sigGenes,expGenes,goAnno,
                          pValThresh = 0.05, plotGO = TRUE,
                          algorithm = "classic")
head(goOutput)
```

Note that in the returned plot, darker colors are more significant. See also that an entire series of GO terms is often calculated as over-represented. If only the very bottom node is truly over-represented, but by a large degree, it can cause the higher nodes (which are also assigned to the genes of the lower node) to appear over-represented as well.

Imagine, for example, that all 10 genes annotated to a low-level GO term (say "negative regulation of DNA ligation") are DE. This GO term is clearly over-represented. But, imagine that in the next level up ("regulation of DNA ligation"), there are 5 additional genes (perhaps "positive" regulators), and none of them are DE. In most data sets, 10 out of 15 genes will still be scored as significantly over-represented, even though it is only the "negative" regulators that we are interested in.

This non-indepenence makes it difficult to know how to interpret each of the significant terms, and leads to large philosophical questions on the appropriate multiple-testing correction. However, the `topGO` authors have come up with a solution to this problem. Essentially, they test to see which node in a chain is the most over-represented, and adjust their p-values to reflect that. The math is quite complex, and beyond the scope of this tutorial; however, the paper describing

the approach is listed in Further Reading below. In the end, it gives us a smaller list of GO terms that is more focused on the real source of differnces. We can see it in action by comparing our results from each method directly:

```
## Run GO analysis with weight algorithm
goOutputWeight <- runGOAnalysis(sigGenes,expGenes,goAnno,
                                pValThresh = 0.05, plotGO = TRUE,
                                algorithm = "weight")
head(goOutputWeight)
```

You can flip back and forth between the plots using the blue arrows at the top of the plot device. Doing this, you should be able to quickly see the difference between the two algorithms. You can also view the full data sets to see that many more GO terms were included in the classic approach, though they may not be more informative.

A few notes on significance testing are needed here. The authors of the weight algorithm argue that their approach implicitly accounts for both multiple testing and the dependency of the GO graph, and so argue that the returned p-value does not need correcting. It is up to you, and your research question, to decide where to draw the threshold for significance. As with the GO annotations themselves, the trick is to find the right balance of conservative approach to neither over- nor under- interpret your data.

5.6. Where to go for help

For more information on what is possible in `topGO` check out the help pages, or visit their website (listed in further reading). In addition, check out statistics sources (textbooks, professor, wikipedia, etc.) for more information on the underlying tests being performed.

5.7. Further Reading

More information on the `topGO` package, including the full package vignette and a link to the full paper (available open access at the link below) can be found at:

- http://www.bioconductor.org/packages/release/bioc/html/topGO.html
- http://bioinformatics.oxfordjournals.org/content/22/13/1600.short

Chapter 6:
Variant Detection

6.1. Background

Everything before this chapter could have been done using microarrays. There is debate in the literature about when RNA-seq and microarrays are more accurate; however, there is one area in which there is a clear winner. RNA-seq analysis adds the ability to detect genetic differences. This added benefit provides a whole new world to explore.

This chapter will work through the very basic levels of genetic analysis of RNAseq data. Where you take it from there is up to you. With the small sample sizes of most of the groups here, population genomic tests may be out of reach. It may, however, be possible to focus on genes of interest (e.g., those in known pathways or those identified as differentially expressed) to add an intriguing layer to your classroom analysis. In addition, with model systems, the role of many of these genetic variants may be known already. For now, we will simply focus on the tools that are available, and the analyses that are possible. There are many different options for the tools to use, but many rely on the same bowtie outputs generated for expression analysis. Today we will use just one of the downstream options.

This chapter is likely to contain less detail than others on the process, and I strongly encourage you to read the documentation included in "Further Reading" before applying these approaches to your project. Like differential expression, genetic variant detection is suffering from a crisis of statistics. The datasets are far larger than can be handled by many of the early variant detectors (or population genetics methods), and new methods are lagging behind the increases in sequence production. The biggest problem is differentiating between sequencing/alignment errors and true variants.

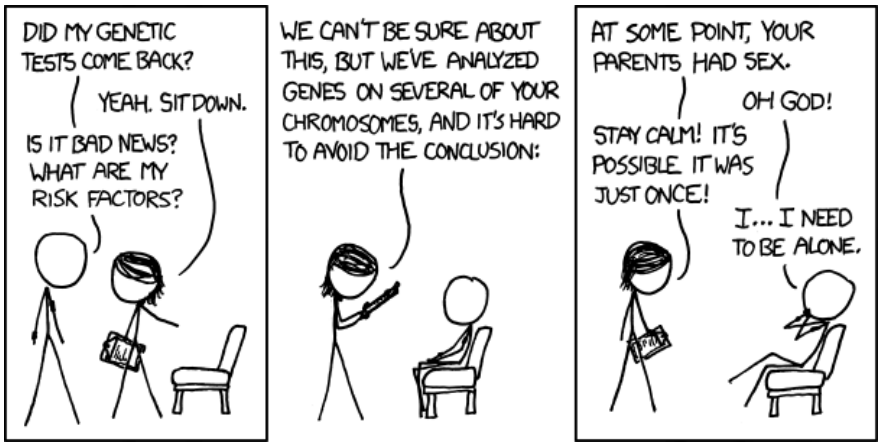

Figure 6.1: xkcd.com/1168 (Creative Commons Licencse)

6.2. Chapter goals

- Learn the basics of SNP detection and ideas for where to go next
- Understand the basic principles managing large datasets

- Understand the statistical principals of large datasets

6.3. Set up the software

We are going to be using a tool named VarScan in this chapter, which requires the samtools package as support. Samtools is available as a module on many computer systems, and VarScan is a simple java file (like Trimmomatic). Recall that we have previously installed and loaded java, but if you missed that step, you will need java to run the below (see section 2.4). To install these tools, use the following commands, which first download VarScan, then load samtoools, then set samtools to load automatically.

```
cd ~/opt
wget http://downloads.sourceforge.net/project/varscan/VarScan.v2.3.6.jar

module load samtools
echo 'module load samtools' >> ~/.modules
```

6.4. Prepare the alignment file

For this example, we will create an alignment file from a small subset of our read files. To run this for a more complete analysis, it would be necessary to run both this step (preparing the mpileup file) and the next step (analyzing the output) in a script submitted via `qsub`.

Move into the directory with the sample sequence files, then each of you will analyze a couple of the samples (the sample you were assigned previously, plus one other of your choice). Note that we are using the same reference as what we used for REM, but we need to call the FASTA file directly. Below, file is the output from the `rsem-calculate-expression` run in section 3.6, and file2 is any additional file (i.e., one created by another student) from the directory.

```
cd /N/dc2/projects/GCAT/workshop/seqData
samtools mpileup -f transcriptome/gcatRef.transcripts.fa   \
                 align/file.transcript.sorted.bam  \
                 align/file2.transcript.sorted.bam \
                 > tempData_yourName.mpileup
```

This creates an encoded file containing a lot of information about the aligned sequence file. Use `less` to take a look at it. As you can see, we need something to help us interpret it. That is where VarScan comes in.

6.5. Run VarScan

VarScan is written as a java file, which means we call it by calling java for the command of interest and passing the options we want. Below, we call the mpileup file we just created, pass it a threshold p-value of interest, then use the redirect ('>') to save the output. We are accepting all of the defaults from `mpileup2snp`, but we could set several options to control the exact output. Look at the documentation linked in "Further Reading" (section 6.8) for more details and to understand the output.

```
java -jar ~/opt/VarScan.v2.3.6.jar mpileup2snp \
          tempData_yourName.mpileup \
          -p-value 0.01 > tempVarScan_yourName.txt
```

Use `less` to look at the file. Remember here that we only have a small number of reads and samples, so the data are likely to be less than complete. To explore these data more fully, download the output file (using `scp`) and load it into R using `read.table()`. From here, there are several functions in the `rnaseqWrapper` package that might help you explore the data, though many of these assume that the reference contains only an in-frame coding sequence (e.g., predicted cds), which we don't have for these data (the fasta reference is not necessarily in-frame, and frequently includes un-translated regions). However, for your projects, these may be useful:

parseVarScan Separates the columns of varScan output format, which improves some usability. This step is not needed for downstream use, but may still prove helpful for other applications.

kaksFromVariants Calculate Ka/Ks ratios using the identified variants

nSynNonSites Calculate the number of synonymous and non-synonymous sites in genes from a reference. Useful to complement the Ka/Ks output for genes with no identified variants.

determineSynonymous Determines whether each variant is non-synonymous or synonymous compared to the reference position. Also calculates dN/dS (wihout respect to sites).

calculateThirdPosBias Calculate the portion of variants (in each gene) at each codon position. A nominal proxy for Ka/Ks and dN/dS, if needed. When no reference is used, it assumes the most common variant position is the third position.

6.6. Run a more complete analysis

Now, to put all of these together, I ran the commands below to create a single file to analyze. Note that the " * " indicates any match, so, it will match all of the temp sequence files in the directory. This is just one more reason why you should be extra careful to be consistent in file naming conventions.

Do NOT run this:

```
samtools mpileup -f transcriptome/gcatRef.transcripts.fa \
                 align/*.transcript.sorted.bam > tempData.mpileup

java -jar ~/opt/VarScan.v2.3.6.jar mpileup2snp \
          tempData.mpileup -p-value 0.01 > tempVarScan_defaultOptions.txt
```

This created a file with all of the samples analyzed, and may be more interesting to analyze.

6.7. Where to go for help

My background is (definitely) not in population genetics, and my research has focused largely on gene expression analysis. However, I have gotten good at reading programmatic documentation, and in time you will too. So, when you start your project, start with the manuals and documentation, and go from there. In addition, there are a lot of really helpful examples online, including answers to other people that faced issues similar to what we are likely to encounter.

6.8. Further Reading

More information related to these topics can be found in:

- http://varscan.sourceforge.net/using-varscan.html
- http://en.wikipedia.org/wiki/Nucleic_acid_notation#IUPAC_notation

Chapter 7:
De Novo Assembly

7.1. Background

One of the most computationally difficult processes in RNA-seq is aligning all of the reads together into transcripts. This can be eased by aligning them against a reference genome or transcriptome, but sometimes references are not available. Only a small number of eukaryotes, and even fewer animals, have had their genomes sequenced. This means that analyzing any non-model system will often require assembling the sequences with just themselves (*de novo*).

This process is a major computational task, often requiring days or even weeks to run. Quite simply: this is not something you want to run on your local machine. Even many online tools don't allow users to run assembly due to the computational cost.

There are a growing number of assembly programs, and each has unique benefits and drawbacks. Below, we will just touch on some of the basics of these assemblers and set up a de novo assembly to run with the cleaned reads that everyone has worked with. We will likely not cover this section in the workshop, but it is here for your edification and benefit.

This module is going to focus on running a simple RNA assembly from the command line, We will walk through each step in the process, leading to a single job with input from all of the samples. The focus in these written instructions will be on running the program, rather than on what the program is doing. For further information on what each assembler does, why you might choose a particular assembler, and what parameters may need tweaking, please see the documentation for these projects, and the links below (see section 7.6).

Figure 7.1: Drew Sheneman – The Newark Star Ledger; available online at genomicglossaries.com/presentation/SLAgenomics.asp

7.2. Chapter goals

- Learn the basics of starting an RNA assembly
- Understand the basics of Linux command line scripting
- Understand the basics of what Trinity, or a similar short read assembler, does

7.3. Getting the data together

For this assembly, we will need to get all of the sequence data combined into a single (very large) file. For that, we will use the `cat` (concatenate) command, and create a new file for each the left and right reads. The code below will combine the listed files together. Replace the dots and the last file with the remaining files to concatenate, and this will create a single file for all of the reads. If you trimmed your sequence data, you should use the trimmed versions here. Low quality sequence data is a much larger hindrance to assembly than it is to gene expression analysis.

```
Do NOT run this:
cd /N/dc2/projects/GCAT/workshop/seqData
cat fileNameA fileNameB fileNameC > allSamples.fq
```

This command combines all of the named files together, and then writes them to a new file. Be careful with the single ">" as it will overwrite a file of the same name. Using ">>" will append the new information to an existing file, but can cause problems with accidentally duplicating information in a file unless care is taken. That was a lot of typing, and introduces several opportunities for error. The problem only gets worse with more files (these sample data came from a set with 90 samples). Luckily there is a ~~lazier~~ more efficient and safer solution using the wild card character (*):

```
Only one student should run this:
cd /N/dc2/projects/GCAT/workshop/seqData
cat *.fq  > allSamples.fq
```

Which will match any of the files ending with ".fq" and put them together in a single script.

7.4. Set up the assembly script

Copy one of your old `qsub` scripts, then open it with nano (or, edit it on your machine, and then `scp` it up). Set the required memory to 10 GB (`vmem=10gb`), and the time to 6 hours (`walltime=6:00:00`). If you are doing this for your own data, you will want much larger memory (e.g. 300gb) and time (e.g. 144 hours). Yes, assembly really can take that long; sometimes even longer, especially if you have more sequence data. This is why you need dedicated computing resources to run this, unless you have a desktop PC with 300 GB of RAM that you don't need for the next week.

Now, we need to add the programs that we will be running. Luckily for us, all of the programs we need (Trinity, samtools, java, and bowtie) are already available on many systems (including Mason). A current list of software available can be found by running the command "`module avail`". To load these programs before running Trinity, add the following lines to your script:

```
Put this in your new script file
module load java
module load samtools/1.2
module load trinityrnaseq
```

Note that you can also run this from the command line, if you want to use Trinity directly from the command line; though the time limits mean you are unlikely to be able to accomplish anything. If you want to load Trinity (or any other module) every time you log on to the computer system, you can add the above line to the file ".modules" in your home directory.

Next, set the working directory for the script to the directory holding our sequence data.

```
Put this in your new script file
cd /N/dc2/projects/GCAT/workshop/seqData
```

We are now ready to add the call to Trinity itself to the script. There are many, many options for Trinity, but we will stick with the simplest of them for this assembly. For more details on options, look at the web documentation for Trinity linked below (section 7.6). Trinity is a Perl

script, which means that it calls another programming language (which you don't need to learn) in order to execute the commands. The available options, and a usage example, can be found by calling "`Trinity --help | less`" (note that this "pipes" the help through `less` to make it easier to scroll). Below are some very basic approaches, which should give you the idea. Add this to your script:

```
 Put this in your new script file
Trinity.pl --seqType fq --max_memory 10G --CPU 1 \
           --output GCAT_Trinity --single allSamples.fq
```

These commands call Trinity. It sets the "`seqType`" to fastq format. Allows the computer to use up to 10 GB of RAM (also needs to be set at the top of the script, and should be changed for your actual data) on one CPU. It will create a new directory named "GCAT_Trinity" for the output, and will use the sequence file we created just above. This one line will run the full assembly for us. Finally, as before, (though here only one student will submit the job) set the job to run with `qsub`, and wait.

7.5. Where to go for help

Assembly is one of the largest challenges in bioinformatics right now. Fortunately, many new approaches are emerging, based on both theoretical and practical exploration. Unfortunately, that means that by the time you get around to looking for help, everything will have changed since I wrote this. Using this module as a starting point will likely get you very far, as Trinity is one of the better assemblers at the moment. If (read: when) you run into problems, ask around to see if it is a common problem, and test out your Google-fu to see if others have already solved it.

7.6. Further Reading

More information related to this topic can be found at:

- trinityrnaseq.sourceforge.net/

Chapter 8:
Simple in-class options

8.1. Background

The ways we have covered so far, focusing on R and the Linux command-line, are the most robust way to do RNAseq analysis, especially for research purposes. However, if you have a four-hour lab available to cover "bioinformatics" for freshman, spending your time teaching them R and Linux may not be fruitful. (Aside: If you have longer, and your goal it to train students to actually do the analysis, it is almost certainly worth your time.) In these cases, you may wish to opt for a simpler approach, with click-able menus, and fewer places for students to wander off-track. In this chapter, I will present a (very) brief introduction to a few of these options. Each lowers the bar to entry into bioinformatics, but generally at the cost of in-depth understanding, flexibility, power, and/or expense.

Many of these tools also ease the entry to research questions as well. However, while they will get you started, they are rarely enough to finish an analysis, and you will likely (eventually) find yourself pulled back to R and the command line.

Figure 8.1: Staples easy button

8.2. Chapter goals

- Introduce simple alternatives
- Understand the utility and disadvantages of various trade-offs
- Explore ways to expose students to bioinformatics

8.3. Galaxy

Galaxy is one of several implementations of a "workflow system." These systems generally include a graphical user interface that makes it easier for those new to bioinformatic analyses (such as students in a class) to begin making progress. The available programs are limited, and default parameters are often set to ease the load on the computers. This makes completing tasks easier, but leaves many decisions out of the investigator or students' hands. The instructions here

are specifically for Galaxy, but common sense and Google will allow you to use these as a guide to enter other systems as well. Because I have more (though still limited) experience with Galaxy, this section will be much more thorough than the others in this chapter.

8.3.1. Get access

There are many implementations of Galaxy that are run by various groups across the country (GCAT-SEEK may be getting one). However, the process for signing up will be similar no matter where you are joining. Google for "galaxy" and look for the Penn State implementation (the url is usegalaxy.org, but Google and one-click will be faster than remembering that). You may wish to bookmark this page to make it easier to get back to it. Click on "user" at the top, and select "Register." Follow the on-screen instructions to get an account. Make sure to remember or save the password you select. Each time you want to sign on, go to the Galaxy main-page. Click "User" at the top, and select "login", and your saved sessions should be available to you.

8.3.2. Load data

There are several ways to load data into Galaxy. Here, we will focus on loading data directly from your computer. However, it is also possible to load in data from other sites on the internet (such as UCSC, BioMart, and several organism specific repositories). All of these methods start with clicking "Get Data" at the top of the left-hand options from any page in Galaxy, then selecting one of the options.

The simplest approach to get data into Galaxy is to load it through the browser. After selecting "Get Data," click "Upload File." Leave the top set to "Auto-detect" to make sure it reads your data appropriately. Browse for your file under "Choose File" and click "Execute." The file should upload and become available in your "History" panel on the right-hand side of the screen. However, this will likely fail for large files.

There are two alternatives to upload larger files. First, you can load the files online somewhere (e.g. Dropbox, Google Drive, etc.) where the file can be downloaded directly. Simply put the file on your sharing site of choice, set the sharing settings to get a url, then paste the url into the box labeled "URL" and click "Execute." You can also upload the files using ftp. Connect to the ftp server listed on the page (it may vary), sign in using your Galaxy credentials, and load the file. Any ftp program will work for this. The file(s) will then be listed on the page – click the box next to the files you want to load and click "Execute."

8.3.3. Install programs

Do nothing. One of the benefits of using this kind of platform is that you (usually) don't have to install programs at all. Everything is already done for you. However, this also means that it is very difficult to add programs, if what you want to use is not present. Further, each instance of Galaxy has a slightly different suite of tools, so it may be worth looking at other options (e.g., Indiana University has a Galaxy instance with a few more tools than the main instance). To run the program you are interested in, find it in the list on the left hand side of the screen, select it, and follow the on-screen prompts.

8.3.4. Where to go for help

Galaxy is set up with a large number of options for help, including a wiki, video tutorials, a "Support" page, and a lot of input from the community. Simply click "Help" at the top, and select one of the options. Alternatively, Googling for "in Galaxy how do I ..." followed by your problem is likely to bring up some great advice as well. When in doubt, reach out to someone that may be able to help you, either directly or by suggesting good terms for your search.

8.4. iPlant

The iPlant collaborative was started in 2008 by the National Science Foundation to develop cyber infrastructure for life science research. It is led by researchers at U. Arizona, the Texas Advanced Computing Center, Cold Spring Harbor Lab, and UNC Wilmington. iPlant offers a set of platforms for next-gen sequence analysis, from graphical user interfaces to packaged command line Linux options. Originally developed for use on plant genomics, the tools are not unique by study system, and so this resource was opened to other species as well. The three tools described below (DNA Subway, Discovery Environment, and Atmosphere) are linked by their use of the "iRods Data Store," a repository for uploaded data, which holds your own data along with publicly available data. For the DNA Subway and Discovery Environment, the on-screen prompts, and linked help, combined with your knowledge from this workshop, should get you well started. For Atmosphere, your Linux knowledge will serve you quite well.

To get access to these tools, go to http://www.iplantcollaborative.org/, the main site describing the iPlant initiative. Select "User Portal", then "Register" and follow the on-screen instructions. In the future, log-in through the same "User Portal" page.

8.4.1. DNA Subway

The simplest of the tools is the DNA Subway, and the "Green line" allows very basic RNAseq analysis. This tool's benefits are also its drawbacks. It is a simple, and very polished, point and click tool through a basic analysis, particularly when using their sample data. This means students don't need to learn, or even be exposed to, R or the Linux command line. However, this also means that there is little flexibility and little interaction, potentially limiting student engagement, while ensuring they don't wander off into the weeds. As of this writing (2014 June), the Green Line is limited to four samples, and only works with eleven model species genomes. This ensures consistent results, but means that your organism of interest may not be available and that the data you are collecting this summer may exceed these limits. In short, this is a great tool for an introduction to the steps of RNAseq analysis, but is not designed for research.

8.4.2. Discovery Environment

The Discovery Environment, on the other hand, is much more similar to Galaxy. The user interface is different, as is the set of tools available. However, many of the programs you may be interested in using will be available, and you can use your own data (via the Data Store) following simple on-screen prompts. Like Galaxy and the DNA Subway, this can ease the transition into bioinformatics for students, though likely will not be able to take you all the way through to your research goals. The major benefit over Galaxy is that, through its use of the data store, it is easier to transfer the data to a more complete system, specifically Atmosphere.

8.4.3. Atmosphere

Atmosphere is, essentially, a cloud-based tool to access pre-built and custom Linux environments (instance) with various programs already installed. Launching an Atmosphere instance will start up a Linux command line or graphical user environment with very good connection to the Data Store, easing file saving/loading problems. The consistent environments made possible by creating (or using) a specific instance, and the connection to the Data Store, will reduce some of the early hurdles to student use. However, while using a created instance is simple, it can be difficult to build one with exactly the tools you want for a class. The set up can be tricky to get just right, and the limits on software space can be problematic if the program you want has many dependencies. For long processes, a separate instance may be required for each step in the analysis.

8.5. Accelerating Comparative Genomics

Accelerating Comparative Genomics (CoGe) is another online tool with a click-able user interface. CoGe is geared towards genomic comparisons (hence the name), but has a few tools which may also be useful for RNAseq analysis. CoGe also integrates with the Data Store, which may make it an especially great tool if you are also interested in using pieces of the iPlant collaborative. I encourage you to look through their help for general information, as well as to refer to the modules written by Dr. Vincent Buonaccorsi for his GCAT-SEEK breakout session on Eukaryotic Genomics: Comparative genomics, for more information on what is possible in CoGe.

There is, however only one tool for RNAseq, described in one of the CoGe youtube videos here: youtu.be/3fNyHGB02dM. While the tutorial takes only a few minutes, please note that they are using an incredibly small data file. In addition, there is no room for modifying parameters, and the information it outputs (on gene models) may not be what you are interested in. As part of the CoGe approach to comparative genomics, this tool is built for gene annotation, and as such, has no support for differential expression. However, it may still be a nice tool to demonstrate some of the basic concepts in short class time or extend your genetic analyses.

8.6. David Functional Annotation Tool

Functional group analysis (described more fully in Chapter 5), requires a fairly substantial amount of preparation work. There are several online tools to ease running GO functional group analysis, including within both iPlant and Galaxy. However, these tools still require many of the same preparation steps, such as obtaining GO annotations. An alternative is to use a web-based tool specifically designed for GO analysis, such as the David Functional Annotation Tool.

The David website (`http://david.abcc.ncifcrf.gov`), allows users to input gene IDs from many model systems, and it then uses available annotations, without the user needing to find them. Someone working with Arabidopsis can, for example, just submit a list of DE genes (based on whatever threshold you are using). They then submit a background gene list, which is most appropriately the genes known to be expressed in the sampled tissue (rather than the DAVID default of all genes in the genome). David will then supply a list of significantly over-represented GO terms and allow for some further analysis.

Several options exist to modify the David output. The user can, for example, select to only analyze GO levels that are appropriate for the functions of interest. This is most often not the most broad and basic levels. Biological or Molecular Process levels 3-7 are often the most informative. In the DAVID tool, the P-value is adjusted using the Benjamini FDR correction by default, but different corrections are available.

This tool still presents challenges to non-model species studies. Because the tool only has access to model system GO annotations, it won't be able to analyze a gene list from other species. However, here, instead of having to run a full GO annotation (as described in Chapter 5), the user would submit the gene identifiers of the best blast hit. David will then analyze those homolog gene identifiers (e.g. Flybase or TAIR gene names) in place of the actual gene names from non-model species. The standard caution remains however, that these annotations should be taken with caution, as the function of a gene may have changed over the evolutionary distance between your organism and the model-species reference.

David does a fantastic job of outputting basic GO analysis information that is well suited for use in a brief class discussion, or when analyzing a model system for which the defaults work. However, as with the other web-based tools we have encountered, there is slightly less flexibility than when working in R or other systems. In addition, if you are generating gene lists in R already, it may be easier to just call a variable name than to copy and paste the results into a new tool and save them outside of R.

8.7. Commercial Software

Up until now, all of the programs that I have described are free and open-source. This means that they are free-as-in-beer (they cost nothing) and free-as-in-speech (you can see all of the code, and do what you want with it). However, not all software falls into this category. There are a number of software packages available for a cost. These systems generally have very clean, attractive, and intuitive graphical user interfaces (GUIs) that make working with them simple. In addition, they generally package all of the tools you may want together; giving you a single installation, rather than the multitude we have used here. If you have the money, and the inclination, such software can make the entry into bioinformatics more pleasant.

These systems come with downsides, including lack of transparency, lack of community support, and cost. First, because the software is proprietary, you don't know exactly what it is doing with your data, which can make interpreting the outcomes very difficult, and makes troubleshooting nearly impossible in some cases. Second, the proprietary nature means that there is, generally, far less community support for the software. This limits the number of help posts you are likely to find, and also means that new features aren't emerging from the field. In contrast, new packages and software are constantly emerging to extend open-source software to accomplish new things (such as the R packages we used here). Finally, these programs can be very expensive. Licenses (which last a limited time) can run well into the thousands of dollars, quickly reaching prohibitive levels for most labs.

That said, many of the users of these commercial programs swear by them. If you haven't noticed yet from the size of my soap box, I am both too cheap and too devoted to open-source to have much experience with these tools. However, below is an incomplete list of software that you may be interested in pursuing.

- Avidis NGS
- CLC Genomics Workbench
- DNASTAR QSeq
- Partek Genomics Suite
- OmicsOffice for NGS
- NextGENe

8.8. Further Reading

For completeness, here is a list of open-source bioinformatics software: http://en.wikipedia.org/wiki/List_of_open-source_bioinformatics_software and a general list of RNAseq tools available, including both open-source and commercial options, sorted by step: http://en.wikipedia.org/wiki/List_of_RNA-Seq_bioinformatics_tools

Chapter 9:

Lab Work

9.1. Background

Most of the work for next-generation sequencing, especially RNA-seq, comes after all of the sequencing is done and centers around four repeated letters in the computer. However, to get to that point (where we have spent the rest of this manual), many laboratory steps must be completed. These steps are not, generally, difficult, but they require a great deal of caution as the finished product (RNA in this case) is often unstable and subject to contamination.

Here, we will walk through the basics of the laboratory steps required for RNA-seq: RNA-extraction, rRNA removal (if necessary), library preparation, and sequencing. Often several of the steps may be performed by a core facility for a fee. This fee is often a convenience fee, though it reduces the need to develop substantial infrastructure for individual investigators, and reduces the likelihood of errors caused by doing something for the first time.

This module will walk through the basics of the lab work necessary for RNA-seq analysis. It will not, however, go into great detail about any of these steps. The simplest approach to each of these steps is to buy a kit and follow those specific directions as exactly as possible. There are a number of less expensive routes, that involve a series of chemicals rather than proprietary kits. These are often much easier to scale for different volumes or applications, but require a bit more up-front expenditure on infrastructure. For this year's workshop, we are lucky enough to be using a protocol, developed by Dr. Arthur Hunt, to make in-expensive librarires ourselves. This protocol will be distributed separately.

Figure 9.1: xkcd.com/699/ (Creative Commons Licencse)

9.2. Chapter goals

- Learn how to extract RNA
- Learn how to prepare a sequencing library
- Make informed decisions about lab techniques and kits
- Understand the process that goes into lab work for RNA-seq

9.3. Sample collection

This process is entirely system, species, and question dependent. There are, however, a few standard rules-of-thumb. First, be fast. Whatever technique you are using, you want it to affect your tissue/cells as little as possible. Therefore, move quickly but not in haste. However, some studies suggest that animal tissues can be processed several hours after death (Cheviron et al., 2011). Second, get things cold (fast). Dry ice, liquid nitrogen, or a -80 C freezer will stop much of the RNA degradation, and can often be used as a preservative even without any sort of buffer (depending on your system). Some alternatives rely on chemical preservation rather than cold temperature. Finally, be consistent between samples. One of the biggest worries is accidentally (or by design) treating samples from experimental groups differently, which could easily cause false differences to be detected. With consistency, there is some leeway from the first two rules. Even if there is some RNA loss, if it is consistent across groups, the stats should catch any problems.

9.4. RNA extraction

RNA extraction for RNA-seq is essentially identical to RNA extraction for other applications, such as microarray processing. Everyone has a preferred method of extraction, generally a kit designed for their species, and most of these methods appear to yield relatively similar results. The basic concept is to rupture the cell, releasing RNA (and other cell contents, such as DNA and protein) in a solution that will protect the RNA from degradation. Most kits are designed to only collect RNA (the RNA sticks to a filter, and is later removed - "eluted") after it is cleaned. Some applications, such as phenol-chloroform extraction with TRIzol, are designed to allow for the collection of genomic DNA, protein, and RNA all a the same time (if desired). This is accomplished by separating the cellular contents into layers that can then be collected, rather than relying on a single filter. A detailed protocol for this is available from me if you are interested.

9.5. rRNA removal

Ribosomal RNA (rRNA) makes up around 80% of cellular RNA, depending on species and estimate. Unfortunately, this means that a large portion of RNA sequenced will be rRNA, which makes it much more difficult to identify the coding sequences of interest. For eukaryotic samples, rRNA removal can be skipped if the library preparation (see next section) uses poly-a isolation. For prokaryotic samples, however, it is necessary to remove rRNA before library prep, unless they are of specific interest to the project.

9.6. Library Preparation

The final step before sequencing is library preparation. RNA must be converted to cDNA and ligated to adapters for sequencing. For most preparations, this will involve breaking the material into smaller pieces and selecting a very specific size fraction. For paired-end sequencing, this

selection is what allows you to know how much space lies between the two sequence pairs – a necessity for accurate assembly and analysis.

Library preparation can require a substantial investment in infrastructure (largely for breaking and size-selecting the RNA and having primers on hand) as well as attention to detail that may be difficult to achieve for students or first - time investigators without substantial oversight. Depending on your available time, infrastructure, guidance, expertise, and money, it is often simpler (and more reliable) and occasionally less expensive, to pay for a core laboratory to perform this step. This reduces errors and allows them to use their infrastructure (often including robots) to reduce time and individual investigatory investment in equipment.

9.7. Where to go for help

The best source for help on lab work is the kit directions and manufacturers themselves. In addition, many common questions have been asked and answered in online forums such as seqanswers.com. A bit of searching may turn up an answer, or at least direct you to somebody (or a forum) that may be able to answer your questions.